연구보고 2004-2

서울대 도서관 정보시스템 유니코드 체계 구축에 관한 연구

서울대학교 중앙도서관

서울대 도서관 정보시스템 유니코드 체계 구축에 관한 연구

연구책임자 : 신 효 필 (서울대학교 언어학과)
공동연구원 : 허 남 진 (서울대학교 철학과)
공동연구원 : 허 성 도 (서울대학교 중어중문학과)
연 구 원 : 김 성 증 (서울대학교 중앙도서관 정보관리과장)
연 구 원 : 황 영 숙 (서울대학교 중앙도서관 정보관리과)
연 구 원 : 김 미 향 (서울대학교 중앙도서관 정보관리과)
연구보조원 : 조 인 식 (서울대학교 언어학과)
연구보조원 : 최 형 주 (서울대학교 인지과학협동과정)

2005. 1.

서울대학교 중앙도서관

국립중앙도서관 출판시도서목록(CIP)

서울대 도서관 정보시스템 유니코드 체계 구축에 관한 연구 / 신효필...[등]연구. -- 서울 : 서울대학교 중앙도서관, 2005
p. ; cm. -- (연구보고 ; 2004-2)

권말부록으로 'ISO 2022에서 사용하는 문자 세트에 대한 용어의 정의들', 'KS C 5601: 1987 문자표', 'CJK Ideograph Area와 - JK Compatibility Ideograph Area간의 변환 테이블 및 변환 출력 프로그램' 수록
참고문헌수록
ISBN 89-956410-2-9
ISBN 89-956410-0-2(세트)

027.611-KDC4
027.7095195-DDC21 CIP2005000547

< 목 차 >

< 표 목차 >

<그림 목차 >

1. 서 론

1.1. 목적 및 필요성

초기 컴퓨터 시스템은 기본적으로 ANSI에서 제정한 ASCII 코드에 기초하여, 영어 알파벳 26자와 약간의 특수기호만을 표상할 수 있었다. 이 시스템들은 연산 속도 및 기억용량의 한계를 가지고 있었고, 동양권 언어들을 전혀 고려하지 않았으므로 이 정도의 성능과 정보 표상방식으로도 충분한 성능을 발휘하였다.

최근에는 정보 시스템의 보급이 일반화되고 인터넷의 발달로 자료의 교환이 국제적으로 확대됨에 따라 표준 문자 코드의 사용은 더욱 중요시되고 있다. 하지만, 기존의 시스템은 몇 가지 어려움에 봉착하게 되었다. 첫째, 동양권의 많은 문자들을 모두 수용하기 어렵고, 둘째, 설령 어떤 표상방식을 고안하더라도 매우 복잡하고, 셋째, 두 번째 방식을 취할 경우 모든 문자를 한꺼번에 표상할 수 없다. 예를 들어, 인터넷에 게시판을 만든다고 가정할 때, 현재의 한 바이트 혹은 이를 이용한 멀티바이트 시스템에서 세계 각국의 사람들이 자국어 문자를 표시할 수 있는 게시판을 인터넷 게시판을 제작하려면, 각 페이지에서 인코딩 방식을 바꾸어야 하므로 그럴 때마다 문자는 다른 방식으로 해석되어 일부 언어의 문자는 깨지고, 전혀 다른 문자로 해석되게 된다. 다시 말해서, 컴퓨터 사용자들의 표현 방식에 대한 욕구는 높아지는 반면, 기존의 시스템으로는 이를 충족시키지 못하는 결과를 초래하게 된 것이다.

이런 문제점을 해결하기 위해 ISO(국제 표준화 기구)와 유니코드 컨소시엄이 협력하여 유니코드를 제정하였다. ISO/IEC 10646에 의해 국제 표준코드로 자리 잡은 유니코드는 세계 모든 문자에 대해서 각각 고유 번호를 할당하므로, 문자를 표현하기 용이하고, 컴퓨터에서 자유로운 표현을 보장할 수 있게 된다. 유니코드 표준은 Apple, HP, IBM, Microsoft, Oracle, SAP, Sun 등과 같은 기업

들이 채택하여 지원하고 있고, 최근 정보 공유 표준으로 자리 잡은 XML(eXtended Markup Language), Java, ECMAScript(Javascript) 등과 같은 최신 시스템 및 언어에서도 채택되고 있다. 또, 인터넷 브라우저 및 다양한 운영체제들이 거의 모두 이를 지원하고 있으며, 이를 개별적으로 지원하는 응용프로그램도 다양하다. 이처럼, 유니코드는 다국어 표현 문제를 해결할 수 있는 유일한 대안으로 떠올랐으며, 미국 의회도서관과 일본 국립정보학연구소(NII) 등 많은 전자도서관들이 유니코드 체계로의 전환을 검토하거나 완료하였다.

본 연구의 목적은 서울대학교 도서관 시스템이 다국어 환경에 능동적으로 대처할 수 있도록 유니코드 체계로의 성공적인 전환을 위한 기반 환경 및 기술 요건을 파악하는 것이다. 현재 국가 표준코드로 쓰이고 있는 KS C 5601:1987을 유니코드로 전환할 때 고려해야 할 사항들이 적지 않다. 도서관 시스템이 탑재된 운영체제, 검색엔진, 데이터베이스, 응용프로그램 등이 유니코드를 지원하는지에 대한 상세한 조사를 해야 하고, 데이터를 전환할 때 발생할 수 있는 문제점에 대한 위험요소들을 미리 분석해야 한다. 또, 다국어 환경에서 사용자들과의 인터페이스 문제도 함께 논의 되어야 한다.

1.2. 내용 및 범위

본 연구는 서울대학교 도서관 시스템을 본격적인 다국어 시스템 환경으로 전환하기 위한 기초 조사로 연구의 내용과 범위는 다음과 같다.

첫째, 기존의 다국어 문자 처리 시스템을 제시하며 유니코드 표준 분석을 통해 기존의 코드 체계에서 유니코드로 전환할 때 발생할 수 있는 문제점들을 파악한다.

둘째, 유니코드 체계를 적용한 국내외의 정보 시스템 및 운영 사례를 조사/분석한다.

셋째, 유니코드를 지원하는 운영체제 및 응용프로그램들을 조사/분석한다.

넷째, 서울대학교 도서관의 서지 및 시스템 현황 등을 기초로 하여 유니코드로의 전환에서 고려해야 할 사항과 그 비용 및 일정 등에 관해 제안을 한다. 또한 현재의 도서관 현황뿐만 아니라 앞으로 서울대학교 도서관에 통합될 여러 자료들의 특성까지도 함께 고려한다.

1.3. 활용방안 및 기대효과

본 연구를 통해 유니코드에 대한 올바른 이해를 할 수 있을 것이고, 다국어 문자 국제 표준 환경에 대응하기 위한 기초 자료로 이용할 수 있을 것이다. 국내외의 사례 분석을 통해 유니코드 전환 시에 고려해야 할 대처 방안들을 모색하고, 서울대학교 도서관 시스템의 유니코드 체계로의 전환에 수반되는 변경 요구사항을 토대로 운영체제, 데이터베이스, 응용 어플리케이션 변경 및 신규 구축에 대한 효과적인 대비가 가능하다.

2. 다국어 문자 표현 방법

2.1. 기존의 다국어 문자 표현 방법

본 장에서는 문자를 컴퓨터에 표현하기 위하여 제정한 표준 문자코드들의 기본 특징을 살펴보고 이러한 표준 문자코드들이 다국어 환경에서 어떤 제약점을 갖게 되는지 기술한다. 그리고 유니코드 문자세트에 대한 기본 구성과 문자세트의 인코딩 방식에 대해 설명하고 유니코드의 제한점을 기술한다.

2.1.1. ASCII 코드

ASCII는 American Standard Code for Information Interchange의 약자로 근대 영어와 다른 서구 유럽 언어들에서 사용되는 로마 알파벳에 기초한 문자 집합과 문자 인코딩 방식으로, 텍스트를 표현하기 위한 컴퓨터와 통신 장비와, 제어장치들에서 가장 많이 사용되고 있다.

다른 문자 표현 코드들과 마찬가지로, ASCII는 디지털 비트 패턴(digital bit pattern)과 문자(symbol/glyph)들 간의 대응관계를 규정하고 있어서, 이를 이용하여 디지털 장비들이 상호 통신하고, 문자에 기반을 둔 정보들을 상호 교환할 수 있다. ASCII 문자 인코딩 혹은 그 확장들은 개인용 컴퓨터와 워크스테이션과 같은 대부분의 컴퓨터에서 사용되고 있다.

ASCII는, 엄격히 말해서, 문자를 표현하기 위해 7비트 코드만을 사용한다. ASCII가 처음 제정되던 당시에는 많은 컴퓨터들이 8비트를 최소 정보 단위 유닛으로(bytes, 또는 더욱 정확히 표현하면 octets) 사용하고 있었다. 8번째 비트는 보통 통신 라인에서 에러를 확인하기 위해 사용되는 패리티 비트(parity bit)로 사용된다. 패리티를 사용하지 않았던 많은 머신들은 8번째 비트를 보통 0로 채워 놓았지만, PRIMOS를 구동시키는 PRIME과 같은 몇몇 시스템들은

ASCII 문자의 8번째 비트를 1로 채워 놓았다.

ASCII는 ANSI의 전신인 ASA(American Standards Association)에 의해 1963년에 처음 제정되었다. ASCII-1963은 소문자들이 없었고, 캐럿 문자(^) 대신에 위쪽 화살표(↑)를 사용하고, 밑줄문자(_) 대신 왼쪽화살표(←)를 사용하는 등 몇 가지 차이들이 있었다. 1967년에 소문자가 추가되었고, 몇 가지 제어 문자들의 이름을 변경하였으며, ACK과 ESC의 두 제어 문자들을 소문자 영역에서 제어 문자 영역으로 이동시켰다. 많은 ASCII 변형들(variants)이 있지만, 가장 널리 사용되는 것은 ANSI X3.4-1986이며, 이는 ECMA-6, ISO/IEC 646 : 1991 International Reference Version, ITU-T Recommendation T.50(09/92), RFC 20 등에서 표준으로 받아들여졌다. 또한, Unicode에서도 최하위 128 문자로 포함되어 있을 정도로 ASCII는 널리 보급된 성공적인 소프트웨어 표준으로 받아들여진다.

역사적으로, ASCII는 전신 코드(telegraphic codes)로 개발되었고, Bell data services에 의해서 선전된 7비트 전신프린터로서 최초로 상업적으로 이용되었다. Bell 시스템은 초기의 5비트 Baudot 전신프린터 코드에 문장 부호들과 소문자를 더한 Fieldata project에서 나온 6비트 코드를 사용할 계획이었다. 하지만, ASCII를 개발하고 있던 ASA 분과 위원회를 참여시키게 되었다. 과거 전신 코드와 비교했을 때, 제안된 Bell 코드와 ASCII는 편리한 알파벳 정렬을 위해서 재배열되었고, 전신프린터 외의 다른 장비들을 위한 요소들이 추가되었다. 또한, ESCape 문자, 백슬래시 문자(backslash), 물결괄호 문자(curly bracket) 등을 포함하는 'ESCape sequence'들도 포함시켰다.

ASCII에서 처음 32개 코드들은 제어 문자로 예약되어 있다. 제어 문자란, 정보를 전달하기 위해 만들어진 것이 아니라, ASCII를 이용하는 프린터 등과 같

은 장비들을 제어하기 위해 고안되었다. 많은 ASCII 제어 문자들은 데이터 패킷들을 표시하거나, 데이터 전송 규약(예를 들어, ENQuiry, ACKnowledge, Negative AcKnowledge, Start Of Header, Start Of Text, End Of Text 등)을 제어한다.

Bin	Dec	Hex	Abb	Printable Representation	Keyboard Access	Name/Meaning
0000 0000	0	00	NUL	NUL	^@	Null character
0000 0001	1	01	SOH	SOH	^A	Start of Header
0000 0010	2	02	STX	STX	^B	Start of Text
0000 0011	3	03	ETX	ETX	^C	End of Text
0000 0100	4	04	EOT	EOT	^D	End of Transmission
0000 0101	5	05	ENQ	ENQ	^E	Enquiry
0000 0110	6	06	ACK	ACK	^F	Acknowledgement
0000 0111	7	07	BEL	BEL	^G	Bell
0000 1000	8	08	BS	BS	^H	Backspace
0000 1001	9	09	HT	HT	^I	Horizontal Tab
0000 1010	10	0A	LF	LF	^J	Line feed
0000 1011	11	0B	VT	VT	^K	Vertical Tab
0000 1100	12	0C	FF	FF	^L	Form feed
0000 1101	13	0D	CR	CR	^M	Carriage return
0000 1110	14	0E	SO	SO	^N	Shift Out
0000 1111	15	0F	SI	SI	^O	Shift In
0001 0000	16	10	DLE	DLE	^P	Data Link Escape
0001 0001	17	11	DC1	DC1	^Q	Device Control 1 — oft. XON
0001 0010	18	12	DC2	DC2	^R	Device Control 2
0001 0011	19	13	DC3	DC3	^S	Device Control 3 — oft. XOFF
0001 0100	20	14	DC4	DC4	^T	Device Control 4
0001 0101	21	15	NAK	NAK	^U	Negative Acknowledgement
0001 0110	22	16	SYN	SYN	^V	Synchronous Idle
0001 0111	23	17	ETB	ETB	^W	End of Trans. Block
0001 1000	24	18	CAN	CAN	^X	Cancel

0001 1001	25	19	EM	EM	^Y	End of Medium
0001 1010	26	1A	SUB	SUB	^Z	Substitute
0001 1011	27	1B	ESC	ESC	^[or ESC	Escape
0001 1100	28	1C	FS	FS	^\	File Separator
0001 1101	29	1D	GS	GS	^]	Group Separator
0001 1110	30	1E	RS	RS	^^	Record Separator
0001 1111	31	1F	US	US	^_	Unit Separator
0111 1111	127	7F	DEL	DEL	DEL or Backspace	Delete

표 1. ASCII의 32개 제어문자들

32번 코드는 빈칸 문자이고, 33번에서 126번까지는 문자, 숫자, 문장부호, 기타 기호등과 같은 인쇄가능한 문자들이다.

Binary	Decimal	Hex	Graphic	Binary	Decimal	Hex	Graphic	Binary	Decimal	Hex	Graphic
0010 0000	32	20	(blank)	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	S	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t

0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C		0101 1100	92	5C	\	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	_				

표 2. ASCII의 인쇄가능한 문자들

컴퓨터 기술이 보급되면서, 로마 알파벳을 사용하는 비영어권 언어들을 쉽게 표현하기 위해 ASCII의 수정판들이 기업들과 표준화기구들에 의해 개발되었고, 이들 중 몇몇은 ASCII 확장들로 간주될 수 있다. ASCII 확장이라는 용어는 7비트 영역에 있는 ASCII 문자 맵을 유지하지 않는 모든 수정판들을 포함하는 것과 종종 혼동되기도 한다.

ISO 646(1972)은 영어 편중 성향을 고치기 위한 첫 번째 시도였지만, 7비트 문자셋을 유지하였기 때문에 호환성 문제를 야기시켰다. 부가적인 코드가 추가되지는 않았지만, 이 언어 종속적인 수정판들에서 일부 코드는 재배열되었다. 따라서, 어떤 수정판인지 알기 못하면 문자를 알 수 없었고, 문자 처리 시스템은 한 가지 수정판만을 다룰 수 있었다.

결국, 기존 7비트 정보대역을 벗어나서, 8번째 비트에 정보를 기록하여 새로운 128개의 문자 코드를 추가하게 되었다. 예를 들어, IBM는 8비트 코드 페이지들을 개발하였는데, 여기서 제어문자들을 그래픽 심벌로 바꾸고, 추가적인 그래픽 문자들을 상위 128 바이트에 배당하였다. 이런 코드 페이지들은 IBM PC

제조사들에 의해 지원되었고, DOS와 같은 운영체제에서도 지원된다.

ISO/IEC 8859와 같은 8비트 표준안들이 ASCII의 진정한 확장안인데, 이들은 기존 문자 배열을 그대로 보존하고, 7비트 영역을 넘어서는 추가 값들만을 추가하였다. 이런 방식을 통해, 더 넓은 영역의 언어들이 표현 가능하게 되었다. 하지만, 여전히 호환성과 제약들에 의해 곤란을 겪고 있지만, 지금까지도 ISO/IEC 8859-1과 기존 7비트 ASCII는 가장 많이 사용되는 문자 인코딩 방식들이다.

2.1.2. ISO 8859

ISO 8859는 공식적으로는 ISO/IEC 8859로 불리며, 컴퓨터 사용자를 위한 8비트 문자 인코딩을 위한 ISO와 IEC의 공동 표준안이다. 이 표준안은 알려진 부분에 따라 ISO/IEC 8859-1, ISO/IEC 8859-2 등과 같이 숫자로 구분되며, 각각은 비공식적으로 그 자체가 표준으로 지시될 수 있다. 현재는 총 15개의 부분들로 구성되어 있다.

96개의 인쇄 가능한 ASCII 문자들만으로도 근대 영어의 정보들을 교환하는데 충분하지만, 로마 알파벳을 사용하는 대부분의 다른 언어들은 β(독일어), □(스웨덴 및 북유럽 언어들)과 같이, ASCII가 다룰 수 없는 추가적인 문자들을 가지고 있다. ISO 8859는 8번째 비트를 이용, 추가적인 128개의 문자들을 위한 위치를 배당하여 이런 문제를 수정하려고 하였다. 하지만, 단일 8비트 문자 인코딩에 맞는 것보다 더 많은 문자들이 필요해져서, 여러 개의 매핑이 개발되었는데, 라틴 문자들을 포함하기 위해서 적어도 10개를 포함하였다.

ISO 8859 표준은 조판을 위해서가 아니라, 신뢰할 수 있는 정보 교환을 위해

서 설계 되었다. 이 표준안은 고품질의 조판에 필요한 문자들을 생략하였다. 결국, 고품질의 조판 시스템들은 ASCII와 ISO 8859 표준안들 위에 종종 독자적인 확장안을 이용하거나, 또는 유니코드 시스템을 이용한다.

대부분의 ISO 8859 인코딩들은 다양한 유럽 언어들에 필요한 구별기호(diacritic)들을 제공하고, 다른 인코딩들은 비-로마 알파벳들(그리스어, 키릴문자, 히브리어, 태국어)들을 제공한다. 하지만, 표준안은 동아시아언어들(CJK : Chinese, Japanese, and Korean)의 문자들에 대한 조항을 만들지 않았는데, 이는 한자 표기 시스템(ideographic writing systems)들이 수천 개의 코드 포인트들을 필요로 하기 때문이었다. 베트남어는 라틴문자에 기반을 둔 문자를 사용하지만, 베트남어는 96 위치만으로는 부족하고, 일본어 음절 가나 문자는 96 개 위치에 들어갈 수 있지만 인코딩되지 않았다.

ISO 8859는 다음과 같은 부분으로 구성되어 있다 :

- ISO 8859-1 (*Latin-1* or *Western European*) — perhaps the most widely used part of ISO 8859, covering most Western European languages: Basque, Catalan, Danish, Dutch (partial[1]), English, Faeroese, Finnish (partial[2]), French (partial[2]), German, Icelandic, Irish, Italian, Norwegian, Portuguese, Rhaeto-Romanic, Scottish, Spanish, and Swedish, Eastern European Albanian, as well as the African languages Afrikaans and Swahili. The missing Euro symbol and capital Ÿ are in the revised version ISO 8859-15. The corresponding IANA-approved character set ISO-8859-1 is the default encoding for legacy HTML documents and for documents transmitted via MIME messages, such as HTTP responses when the document's media type is "text" (as in "text/html").

- ISO 8859-2 (*Latin-2* or *Central European*) — supports those Central and Eastern European languages that use a Roman alphabet, including Polish, Czech, Slovak, Slovenian, and Hungarian. The missing Euro symbol can be found in version ISO 8859-16.
- ISO 8859-3 (*Latin-3* or *South European*) — Turkish, Maltese, and Esperanto; largely superseded by ISO 8859-9 for Turkish and Unicode for Esperanto.
- ISO 8859-4 (*Latin-4* or *North European*) — Estonian, Latvian, Lithuanian, Greenlandic, and Sami.
- ISO 8859-5 (*Cyrillic*) — Covers mostly Slavic languages that use a Cyrillic alphabet, including Belarusian, Bulgarian, Macedonian, Russian, Serbian, and Ukrainian.
- ISO 8859-6 (*Arabic*) — Covers the most common Arabic language characters. Doesn't support other languages using the Arabic script.
- ISO 8859-7 (*Greek*) — Covers the modern Greek language (monotonic orthography). Can also be used for Ancient Greek written without accents or in monotonic orthography, but lacks the diacritics for polytonic orthography. These were introduced with Unicode.
- ISO 8859-8 (*Hebrew*) — Covers the modern Hebrew alphabet as used in Israel. In practice two different encodings exist, logical and visual.
- ISO 8859-9 (*Latin-5* or *Turkish*) — Largely the same as ISO 8859-1, replacing the rarely used Icelandic letters with Turkish ones. It is also used for Kurdish.
- ISO 8859-10 (*Latin-6* or *Nordic*) — a rearrangement of Latin-4. Considered more useful for Nordic languages. Baltic languages use

Latin-4 more.

- ISO 8859-11 (*Thai*) — Contains most glyphs needed for the Thai language.
- ISO 8859-12 — was supposed to be Latin-7 and cover Celtic, but this draft was rejected. Numbering continued with -13.
- ISO 8859-13 (*Latin-7* or *Baltic Rim*) — Added some glyphs for Baltic languages which were missing from Latin-4 and Latin-6.
- ISO 8859-14 (*Latin-8* or *Celtic*) — Mostly a rearrangement of the ISO 8859-12 draft. Covers Celtic languages such as Gaelic and the Breton language.
- ISO 8859-15 (*Latin-9*) — a revision of 8859-1 that removes some little-used symbols, replacing them with the Euro symbol € and the letters *Š*, *š*, *Ž*, *ž*, *Œ*, *œ*, and *Ÿ*, which completes the coverage of French and Finnish.
- ISO 8859-16 (*Latin-10* or *South-Eastern European*) — Intended for Albanian, Croatian, Hungarian, Italian, Polish, Romanian and Slovenian, but also Finnish, French, German and Irish Gaelic (new orthography). The focus lies more on letters than symbols. The currency sign is replaced with the Euro symbol.

Binary	Oct	Dec	Hex	1	2	3	4	5	6	7	8	9	10	11	13	14	15	16
10100000	240	160	A0	*NBSP*														
10100001	241	161	A1	¡	Ą	ħ	A	Ė		‘		¡	Ą	ת	”	Ḃ	¡	A
10100010	242	162	A2	¢	˘	˘	к	Ђ		’	¢	¢	Ē	ט	¢	ḃ	¢	ą
10100011	243	163	A3	£	Ł	£	Ŗ	Ґ		£	£	£	Ģ	ע	£	£	£	Ł
10100100	244	164	A4	¤	¤	¤	¤	Ꞓ	¤	€	¤	¤	Ī	פ	¤	Ċ	€	€
10100101	245	165	A5	¥	Ľ		Ī	S		Ð	¥	¥	Ī	פ	„	ė	¥	„
10100110	246	166	A6	¦	Ś	Ĥ	Ļ	I		¦	¦	¦	Ķ	צ	¦	Ḋ	Š	Š
10100111	247	167	A7	§	§	§	§	Ї		§	§	§	§	ג	§	§	§	§
10101000	250	168	A8	¨	¨	¨	¨	J		¨	¨	¨	Ļ	ק	Ø	Ẁ	š	š
10101001	251	169	A9	©	Š	Ì	Š	Љ		©	©	©	Đ	ר	©	©	©	©
10101010	252	170	AA	ª	Ş	Ş	Ê	Њ		.	×	ª	Š	ש	Ŗ	Ẁ	ª	Ş
10101011	253	171	AB	«	Ť	Ğ	Ģ	Ћ		«	«	«	Ŧ	ת	«	đ	«	«
10101100	254	172	AC	¬	Ź	Ĵ	Ŧ	Ќ	˛	¬	¬	¬	Ž	ם	¬	Ẏ	¬	Ź
10101101	255	173	AD	-	-	-	-	-	-	-	-	-	-	ן	-	-	-	-
10101110	256	174	AE	®	Ž		Ž	Ў			®	®	Ū	ף	®	®	®	ź
10101111	257	175	AF		Ż	Ż		Џ		—			Ņ	ף	Æ	Ÿ		Ż
10110000	260	176	B0	°	°	°	°	A		°	°	°	°	ץ	°	Ḟ	°	°
10110001	261	177	B1	±	ą	ħ	ą	Б		±	±	±	ą	ת	±	ḟ	±	±
10110010	262	178	B2	²	˛	²	˛	B		²	²	²	ē	ש	²	Ġ	²	Č
10110011	263	179	B3	³	ł	³	ŗ	Г		³	³	³	ģ	ת	³	ġ	³	ł
10110100	264	180	B4	´	´	´	´	Д		`	´	´	ī	פ	“	Ṁ	Ž	Ž
10110101	265	181	B5	µ	Ĺ	µ	ī	E		^	µ	µ	ī	פ	µ	ṁ	µ	”
10110110	266	182	B6	¶	ś	ĥ	ļ	Ж		A	¶	¶	ķ	ת	¶	¶	¶	¶
10110111	267	183	B7	·	˘	·	˘	З		·	·	·	·	ק	·	Ṗ	·	·
10111000	270	184	B8	¸	¸	¸	¸	И		Ё	¸	¸	ļ	ט	ø	ẁ	ž	ž
10111001	271	185	B9	¹	š	ı	š	Й		H	¹	¹	đ	ש	¹	ṗ	¹	č
10111010	272	186	BA	º	ş	ş	ě	K		I	÷	º	š	ט	ŗ	ẁ	º	ş
10111011	273	187	BB	»	ľ	ğ	ģ	Л	;	»	»	»	t	ט	»	Ṡ	»	»
10111100	274	188	BC	ł	ź	ĵ	t	M		O	ł	ł	ž	ש	ł	ẏ	Œ	Œ
10111101	275	189	BD	ł	˝	ł	Ņ	H		ł	ł	ł	—	ת	ł	Ẅ	œ	œ
10111110	276	190	BE	ł	ž		ž	O		Y	ł	ł	ū	ש	ł	ẁ	Ÿ	Ÿ
10111111	277	191	BF	¿	ż	ż	ŋ	П	ŗ	Ω		¿	ŋ	ש	æ	ṡ	¿	ż
11000000	300	192	C0	À	Ŕ	À	Ā	P		t		À	Ā	ת	Ą	À	À	À
11000001	301	193	C1	Á	Á	Á	Á	C	˛	A		Á	Á	ש	ļ	Á	Á	Á
11000010	302	194	C2	Â	Â	Â	Â	T	ī	B		Â	Â	ת	Ā	Â	Â	Â
11000011	303	195	C3	Ã	Ã		Ã	У	ł	Γ		Ã	Ã	ן	Ć	Ã	Ã	Ã

11000100	304	196	C4	Ä	Ä	Ä	Ä	Ф	ვ	Δ		Ä	Ä	ŋ	Ä	Ä	Ä	Ä
11000101	305	197	C5	Å	Ĺ	Ć	Å	Х	ļ	Е		Å	Å	ŋ	Å	Å	Å	Ć
11000110	306	198	C6	Æ	Ć	Ć	Æ	Ц	ც	Z		Æ	Æ	ŋ	Ę	Æ	Æ	Æ
11000111	307	199	C7	Ç	Ç	Ç	Į	Ч	ı	Н		Ç	Į	ɔ	Ė	Ç	Ç	Ç
11001000	310	200	C8	È	Č	È	Č	Ш	ბ	Θ		È	Č	ო	Č	È	È	È
11001001	311	201	C9	É	É	É	É	Щ	ს	I		É	É	ъ	É	É	É	É
11001010	312	202	CA	Ê	Ę	Ê	Ę	Ъ	ტ	К		Ê	Ę	ო	Ź	Ê	Ê	Ê
11001011	313	203	CB	Ë	Ė	Ë	Ė	Ы	უ	Л		Ë	Ė	ო	Ė	Ė	Ė	Ė
11001100	314	204	CC	Ì	Ě	Ì	Ė	Ь	ფ	М		Ì	Ė	ო	Ģ	Ì	Ì	Ì
11001101	315	205	CD	Í	Í	Í	Í	Э	ქ	N		Í	Í	ъ	Ķ	Í	Í	Í
11001110	316	206	CE	Î	Î	Î	Î	Ю	ღ	Ξ		Î	Î	ъ	Ī	Î	Î	Î
11001111	317	207	CF	Ï	Ď	Ï	Ï	Я	ყ	O		Ï	Ï	ო	Ļ	Ï	Ï	Ï
11010000	320	208	D0	Đ	Đ		Đ	а	ჯ	П		Ģ	Đ	ː	Š	Ŵ	Đ	Đ
11010001	321	209	D1	Ñ	Ń	Ñ	Ņ	б	ჟ	Р		Ñ	Ņ	-	Ń	Ñ	Ñ	Ń
11010010	322	210	D2	Ò	Ň	Ò	Ò	в	ზ			Ò	Ò	ɔ	Ņ	Ò	Ò	Ò
11010011	323	211	D3	Ó	Ó	Ó	Ķ	г	ს	Σ		Ó	Ó	˙ɔ	Ó	Ó	Ó	Ó
11010100	324	212	D4	Ô	Ô	Ô	Ô	д	შ	Т		Ô	Ô	˙	Ō	Ô	Ô	Ô
11010101	325	213	D5	Õ	Õ	Ġ	Õ	е	ს	Y		Õ	Õ	˙	Ō	Õ	Õ	Ó
11010110	326	214	D6	Ö	Ō	Ö	Ö	ж	ტ	Ф		Ö	Ö	˙	Ō	Ö	Ö	Ö
11010111	327	215	D7	×	×	×	×	з	ჩ	Х		×	Ū	˙	×	Ť	×	Ś
11011000	330	216	D8	Ø	Ř	Ġ	Ø	и	ც	Ψ		Ø	Ø	˛	Ų	Ø	Ø	Ú
11011001	331	217	D9	Ù	Ů	Ù	Ų	й	ძ	Ω		Ù	Ų	˛	Ł	Ù	Ù	Ù
11011010	332	218	DA	Ú	Ú	Ú	Ú	к	ჭ	Ī		Ú	Ú	.	Ś	Ú	Ú	Ú
11011011	333	219	DB	Û	Ű	Û	Û	л		Ÿ		Û	Û		Ū	Û	Û	Û
11011100	334	220	DC	Ü	Ū	Ū	Ū	м		ά		Ū	Ū		Ū	Ū	Ū	Ū
11011101	335	221	DD	Ý	Ý	Ü	Ů	н		έ		Ī	Ý		Ż	Ý	Ý	Ę
11011110	336	222	DE	Þ	Ţ	Ŝ	Ū	о		ή		Ş	Þ		Ž	Ŷ	Þ	Ţ
11011111	337	223	DF	ß	ß	ß	ß	п		ί	_	ß	ß	ъ	ß	ß	ß	ß
11100000	340	224	E0	à	ŕ	à	ā	р	‐	ΰ	ж	à	ā	ı	ą	à	à	à
11100001	341	225	E1	á	á	á	á	с	ფ	α	ב	á	á	ш	ļ	á	á	á
11100010	342	226	E2	â	â	â	â	т	ქ	β	ג	â	â	ī	ā	â	â	â
11100011	343	227	E3	ã	ă		ă	у	ღ	γ	ד	ã	ã	ı	ć	ã	ã	ă
11100100	344	228	E4	ä	ä	ä	ä	ф	ყ	δ	ה	ä	ä	ı	ä	ä	ä	ä
11100101	345	229	E5	å	Í	ć	å	х	ჯ	ε	ו	å	å	ı	å	å	å	ć
11100110	346	230	E6	æ	ć	č	æ	ц	ჟ	ζ	ז	æ	æ	ı	ę	æ	æ	æ
11100111	347	231	E7	ç	ç	ç	į	ч	ზ	η	ח	ç	į	˛	ē	ç	ç	ç
11101000	350	232	E8	è	č	è	č	ш	თ	θ	ט	è	č	˙	č	è	è	è
11101001	351	233	E9	é	é	é	é	щ	ი	ι	י	é	é	˝	é	é	é	é

11101010	352	234	EA	ê	ę	ê	ę	ъ	ﻍ	к	ㄱ	ê	ę	˜	ź	ê	ê	ê
11101011	353	235	EB	ë	ë	ë	ë	ы	˙	λ	ɔ	ë	ë	˙	ė	ë	ë	ë
11101100	354	236	EC	ì	ě	ì	è	ь	˛	μ	ㄣ	ì	è	˛	ģ	ì	ì	ì
11101101	355	237	ED	í	í	í	í	э	¸	ν	ם	í	í	¸	ķ	í	í	í
11101110	356	238	EE	î	î	î	î	ю	˝	ξ	ם	î	î	˝	ī	î	î	î
11101111	357	239	EF	ï	ď	ï	ï	я	˛	ο	ן	ï	ï	˛	ļ	ï	ï	ï
11110000	360	240	F0	ð	đ		đ	ŕ	¸	π	ɔ	ğ	ð	˚	š	ŵ	ð	đ
11110001	361	241	F1	ñ	ń	ñ	ŋ	ë	˝	ρ	ם	ñ	ŋ	˚	ń	ñ	ñ	ń
11110010	362	242	F2	ò	ň	ò	ō	ђ	˙	ς	ע	ò	ō	˛	ŋ	ò	ò	ò
11110011	363	243	F3	ó	ó	ó	ķ	ѓ		σ	ך	ó	ó	˛	ó	ó	ó	ó
11110100	364	244	F4	ô	ô	ô	ô	є		τ	ם	ô	ô	˝	ō	ô	ô	ô
11110101	365	245	F5	õ	ő	ģ	õ	ѕ		υ	ף	õ	õ	˝	ō	õ	õ	ő
11110110	366	246	F6	ö	ö	ö	ö	і		φ	ץ	ö	ö	˃	ō	ö	ö	ö
11110111	367	247	F7	÷	÷	÷	÷	ї		χ	ף	÷	ū	˛	÷	į	÷	ś
11111000	370	248	F8	ø	ř	ģ	ø	ј		ψ	ך	ø	ø	˛	ų	ø	ø	ű
11111001	371	249	F9	ù	ů	ù	ų	љ		ω	ץ	ù	ų	˛	ł	ù	ù	ù
11111010	372	250	FA	ú	ú	ú	ú	њ		ι	ת	ú	ú	˃	ś	ú	ú	ú
11111011	373	251	FB	û	ű	û	û	ћ		ϑ		û	û	˛	ū	û	û	û
11111100	374	252	FC	ü	ů	ü	ü	ќ		ο		ü	ü		ū	ü	ü	ü
11111101	375	253	FD	ý	ý	ŭ	ū	§		υ	LRM	ı	ý		ż	ý	ý	ę
11111110	376	254	FE	þ	ţ	ŝ	ū	ў		ώ	RLM	ş	þ		ž	ŷ	þ	ţ
11111111	377	255	FF	ÿ	˙	˙	˙	џ				ÿ	к		’	ÿ	ÿ	ÿ

표 3. ISO-8859의 여러 가지 매핑값들의 목록

2.1.3. ISO 2022

ISO 2022는, 공식적으로는 ISO/IEC 2022로서, 단일 문자 인코딩(a single character encoding)에서 복합 문자 집합(multiple character sets)을 포함하는 기술을 명세한 ISO 표준이다. (ECMA 표준인 ECMA-35와 동일하다.) 각 문자당 8비트를 사용하는 ISO 8859 문자 인코딩과 달리, ISO 2022 인코딩은 각 문자당 8비트 혹은 16비트를 사용하는 가변 길이 인코딩(variable size encoding)을 전형적으로 사용한다. 몇몇 문자 인코딩들은 ISO 2022의 방법을 사용하는데, 대표적인 예로 ISO-2022-JP는 일본어에 대한 문자 인코딩에서 널리 사용되는 방법이다.

그리스어, 러시아어, 아랍어 또는 히브리어 등과 같이 라틴 알파벳에 기반을 두지 않은, 많은 언어와 어족들은 역사적으로 문자집합의 ISO 8859를 포함하는 8-비트 확장ASCII 인코딩으로 컴퓨터에서 표현되어 왔다. 중국어, 일본어, 한국어와 같은 동아시아 문자들은 8비트 컴퓨터에서 표현가능한 문자보다 더 많은 문자들을 이용하므로, 처음에는 언어 종속적인 2바이트 인코딩(language-specific double byte encoding)으로 컴퓨터에서 표현되었다. ISO 2022 문자 인코딩들은 이어서 나오는 문자들에 대한 문자집합을 지시하는 이스케이프 시퀀스(escape sequence)를 포함하고 있다. 이스케이프 시퀀스(escape sequence)들은 ISO에 등록되어 있으며, 종종 ASCII 이스케이프 시퀀스(16진수 1B, 8진수 33)로 시작하는, 세 문자들이다. 자료들을 올바르게 해석하는 것이 가장 최근에 만난 이스케이프 시퀀스에 달려있으므로, 이런 문자 인코딩들은 스캔진행 방향으로 선형적으로 자료들이 처리되어야 한다. ISO 2022 문자 집합들이 여전히 사용되고 있음에도 불구하고, 최신 이메일 소프트웨어는 UTF-8과 같은 유니코드 문자 인코딩들을 사용하는 것으로 변환되고 있다.

2.1.4. KS C 5601:1987

KS C 5601:1987은 94x94의 각 위치(행렬)에 한글 문자를 일정한 순서에 따라 배열해 놓은 문자세트를 의미한다. 한글 코드의 KS 제정에서 완성형이 채택된 것은 내부적으로 한글의 출력이 모아쓰기 형태로 이루어지면서 한자를 섞어서 쓸 수 있어야 한다는 사회적 요구로 조합형을 수용하기가 어려웠기 때문이다. 또 다른 배경은 국가 간의 정보교환을 위한 코드 표준화 과정에서 ISO 2022에서 제정한 코드 체계에 따라 세계 각국의 문자를 처리하는데 기인한다. 이는 1바이트 코드로 한 문자 표현이 불가능한 CJK(Chinese, Japanese, Korean) 문자를 2바이트 코드 영역의 첫 번째 영역에 넣을 수 있도록 영역을 확보해야 했기 때문이다.

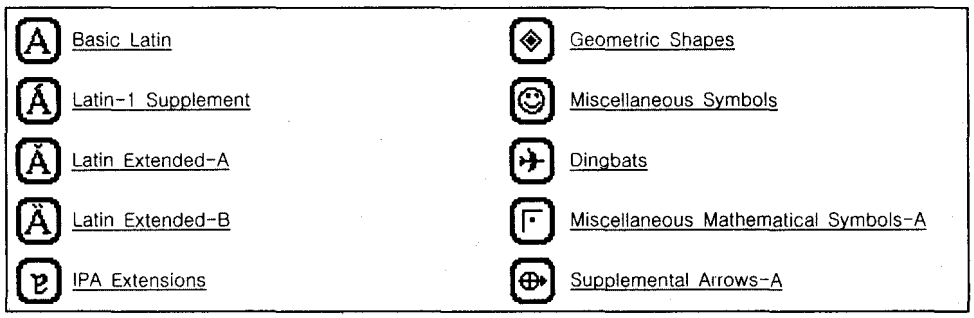
이와 같은 배경에 의해 KS C 5601:1982에 의한 2바이트 조합형 코드가, 1987년에 2바이트 완성형 코드인 정보교환용 부호에 관한 한글 공업 규격으로 새로 바뀌게 된 것이다.

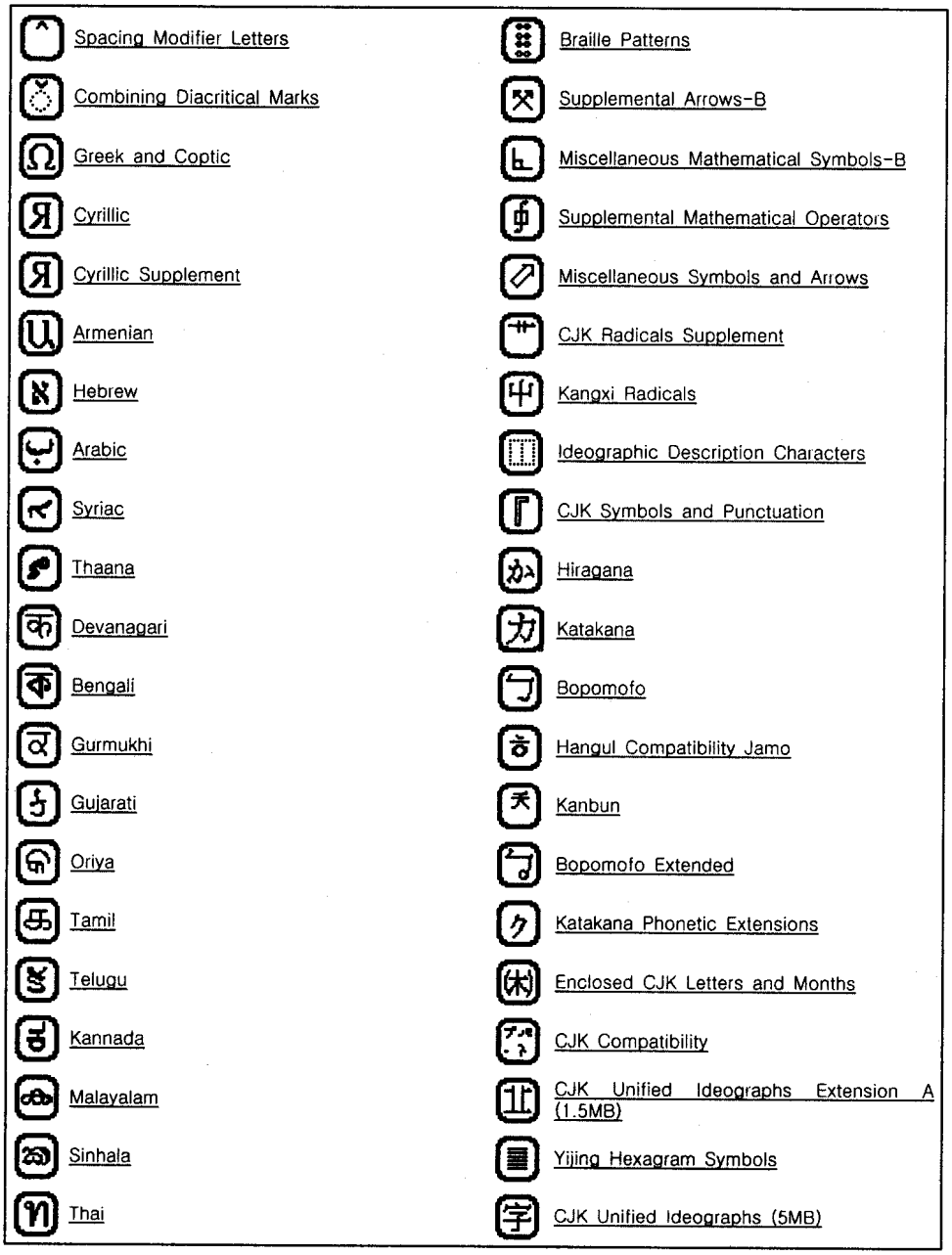
KS C 5601:1987은 완성형 한글 2,350자, 한자 4,888자, 기술/학술기호 등 특수문자 432자, 숫자 30자, 한글 낱자 94자, 로마문자 52자, 그리스 문자 48자, 괘선 조각 68자, 라틴 문자 27자, 일본 문자 169자, 러시아 문자 66자 등 총 8,224자와 기타 사용자 정의 영역으로 한글 96자, 한자 95자 정도를 사용하도록 배정하고 있다. (부록 참조)

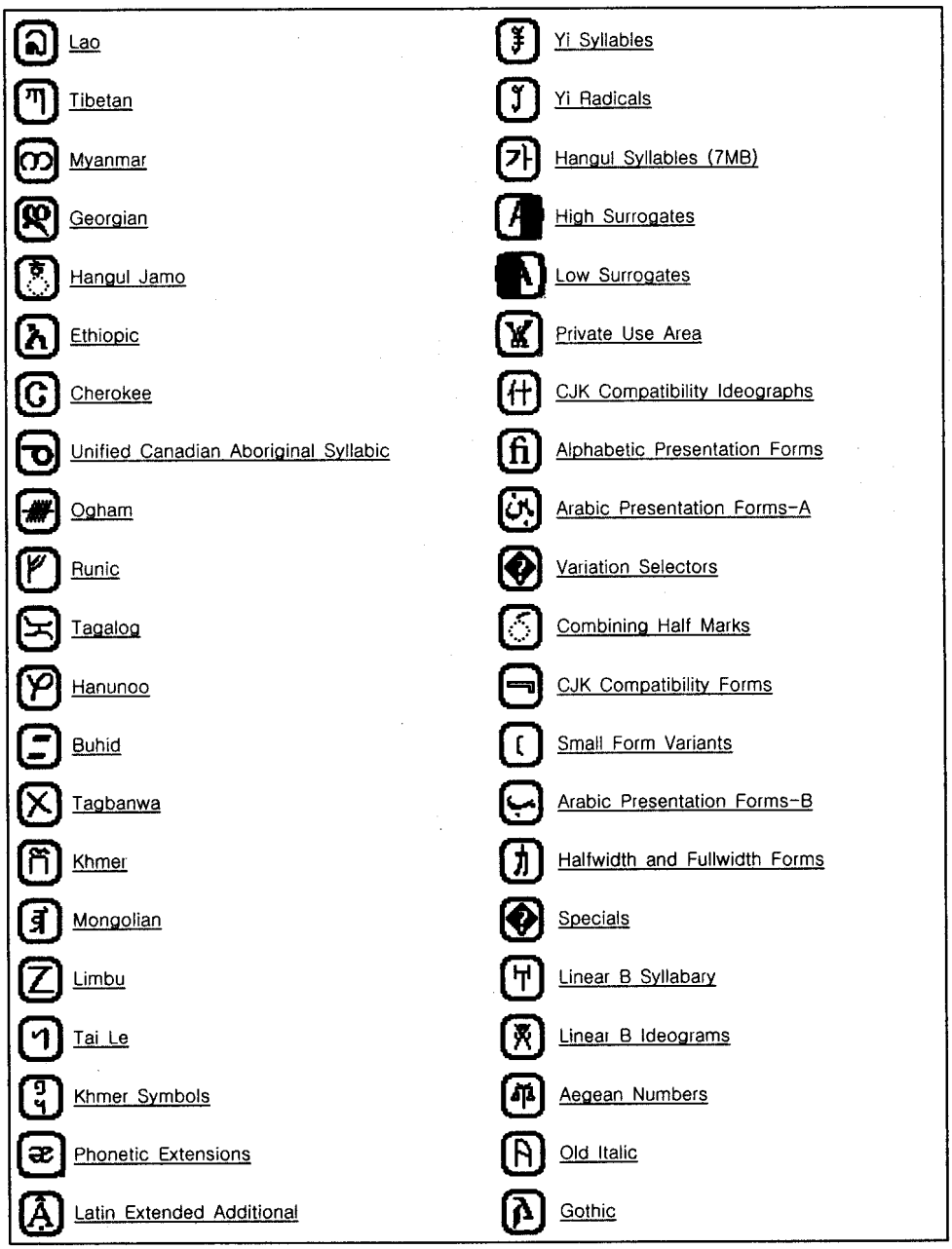
2.2. 유니코드

2.2.1. 개요

유니코드(Unicode)는, 컴퓨터에 저장하려는 모든 문서 텍스트들을 인코딩하기 위한 방법을 제공하기 위한 국제 표준이다. 유니코드는 오늘날 실제 사용되는 모든 문자들과, 학자들만 사용하는 문자들, 또 수학기호, 언어학기호, 프로그래밍 언어 기호 등까지 포함하고 있다.




Spacing Modifier Letters
Combining Diacritical Marks
Greek and Coptic
Cyrillic
Cyrillic Supplement
Armenian
Hebrew
Arabic
Syriac
Thaana
Devanagari
Bengali
Gurmukhi
Gujarati
Oriya
Tamil
Telugu
Kannada
Malayalam
Sinhala
Thai
Braille Patterns
Supplemental Arrows-B
Miscellaneous Mathematical Symbols-B
Supplemental Mathematical Operators
Miscellaneous Symbols and Arrows
CJK Radicals Supplement
Kangxi Radicals
Ideographic Description Characters
CJK Symbols and Punctuation
Hiragana
Katakana
Bopomofo
Hangul Compatibility Jamo
Kanbun
Bopomofo Extended
Katakana Phonetic Extensions
Enclosed CJK Letters and Months
CJK Compatibility
CJK Unified Ideographs Extension A (1.5MB)
Yijing Hexagram Symbols
CJK Unified Ideographs (5MB)


Lao
Tibetan
Myanmar
Georgian
Hangul Jamo
Ethiopic
Cherokee
Unified Canadian Aboriginal Syllabic
Ogham
Runic
Tagalog
Hanunoo
Buhid
Tagbanwa
Khmer
Mongolian
Limbu
Tai Le
Khmer Symbols
Phonetic Extensions
Latin Extended Additional
Yi Syllables
Yi Radicals
Hangul Syllables (7MB)
High Surrogates
Low Surrogates
Private Use Area
CJK Compatibility Ideographs
Alphabetic Presentation Forms
Arabic Presentation Forms-A
Variation Selectors
Combining Half Marks
CJK Compatibility Forms
Small Form Variants
Arabic Presentation Forms-B
Halfwidth and Fullwidth Forms
Specials
Linear B Syllabary
Linear B Ideograms
Aegean Numbers
Old Italic
Gothic

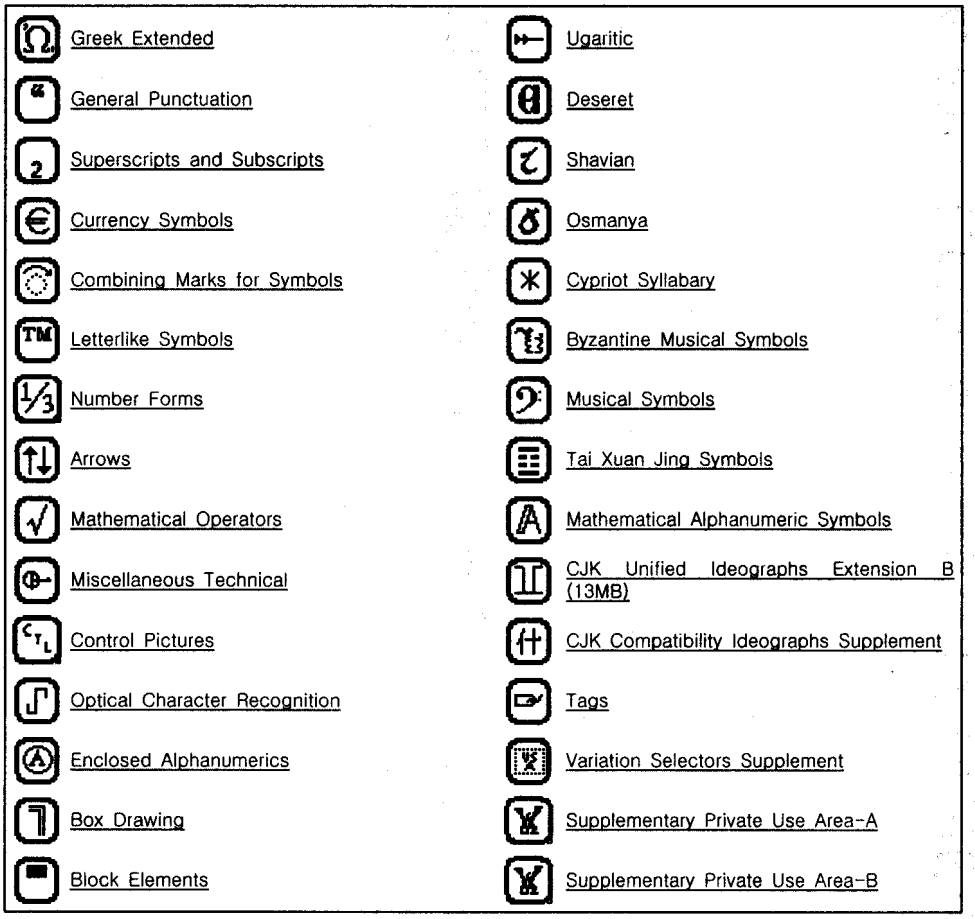


표 4. 유니코드에 포함된 여러 기호들의 목록

2.2.2. 개발 동기

유니코드는 기존의 문자집합들이 가지고 있었던 다국어 환경(multilingual environment)에서의 문제점들을 극복하고, 이들을 대체하기 위해 개발되었다. 유니코드 개발 과정에서 기술적인 문제점들과 제한, 및 비판들이 있었지만, 현재 유니코드는 가장 완벽한 문자집합 및 가장 큰 문자집합 중 하나로 인정받고 있으며, 소프트웨어의 국제화 및 다국어 환경에서 가장 주도적인 인코딩 체계가 되어 가고 있다.

XML과 같은 많은 표준들 및 운영체계와 같은 시스템 소프트웨어들도 텍스트를 표현하는 기저 체계로써 유니코드를 채택하고 있다.

ISO 8859에 의해서 정의된 전통적인 문자 인코딩 방식들은 세계의 다양한 나라들에서 사용되었지만 서로 상호 호환될 수 없었다. 유니코드는 이러한 한계를 극복하려는 분명한 목적을 가지고 개발되었다. 전통적인 문자 인코딩이 가지고 있는 문제점은 이개언어 컴퓨터 처리(bilingual computer processing)는 허용하지만, 다국어 컴퓨터 처리(multilingual computer processing)는 허용하지 않는다는 것이다.

유니코드는 기본 취지에서 기저에 있는 문자들을 직접 인코딩하고, 해당 문자들에 대한 다른 그림문자(variant glyph)는 인코딩하지 않는다. 중국어 문자의 경우에, 어떤 것이 기저 문자이고, 어떤 것이 다른 그림문자이냐에 대해서는 논란의 소지가 있다.

유니코드는 각 그림문자(glyph)가 아닌, 각 문자(character)에게 한 코드 포인트(code point)를 부여하는 것이 목적이다. 좀 더 일반적인 용어로 표현하면, 프린터에서 사용되는 조판 변화에 상관없이, 각 문자(letter)에게 고유번호(unique number)를 부여하는 것이다.

하지만, 이런 단순한 목적은 현존하는 다른 인코딩 시스템들 간에 손실 없는 변환(lossless conversion)을 제공하려할 경우 복잡한 문제를 발생시킨다. 또한, 유니코드 표준은 문자 자질(character properties), 텍스트 정규화(text normalization), 양방향 표시 순서(bidirectional display order)와 같은 많은 연관 내용들을 포함하고 있다.

2.2.3. 매핑 및 인코딩

캘리포니아에 근거한 유니코드 컨소시엄(Unicode Consortium)은 유니코드 표준을 개발한 단체로, 회원 수수료를 지불할 의사가 있는 회사 및 개인에게 개방되어 있다. 유니코드 회원들에는 다양한 소프트웨어 및 하드웨어 회사들이 포함되어 있는데, Apple Computer, Microsoft, IBM, Xerox, HP, Adobe Systems 등도 포함되어 있다.

컨소시엄은 1991년에 "The Unicode Standard"(ISBN 0321185781)를 출판하였고, 여기에 기초하여 표준안을 계속해서 내놓았다. 유니코드는 세계 표준화 기구(ISO : the International Organization for Standardization)와 협력하여, ISO/IEC 10646과 문자 목록을 공유하였다. 유니코드와 ISO/IEC 10646은 문자 인코딩에서는 동일하지만, 유니코드 표준안이 더 많은 세부 규정들을 포함하고 있으며, 두 표준안은 또한 약간 다른 용어를 사용하고 있다.

o 유니코드 개정안 역사

1991 Unicode 1.0

1993 Unicode 1.1

1996 Unicode 2.0

1998 Unicode 2.1

2000 Unicode 3.0

2001 Unicode 3.1

2002 Unicode 3.2

2003 Unicode 4.0

2004 Unicode 4.01

2.2.4. 저장 공간, 전송 및 처리

지금까지 유니코드가 사람들이 사용하는 모든 문자들에 고유번호를 부여하는 수단으로써만 논의되었지만, 이런 고유번호 숫자들이 텍스트 처리에서 저장되는 방식도 또 다른 문제이다. 이런 문제는 서구의 많은 소프트웨어들이 8비트 문자 인코딩을 지원하기 위해 개발되었고, 유니코드 지원이 최근에 매우 천천히 이루어졌기 때문에 발생하였다. 마찬가지로, 아시아에서도 2바이트 문자 인코딩(double-byte character encoding)은 원칙적으로 65,536 문자 이상을 표현할 수 없고, 실제 시스템은 훨씬 적은 문자들을 표현하였다. 따라서, 중국어 학문을 위해서는 매우 부족하였다.

전통적인 8비트 소프트웨어의 내부 논리 연산은 각 문자에 대해 8비트만을 허용하였고, 특별한 처리를 거치지 않고는 256 코드 포인트 이상을 사용하는 것이 불가능했다. 16비트 소프트웨어는 몇 만개의 문자들까지 표현할 수 있었다. 하지만, 유니코드는 90,000개 이상의 인코딩된 문자들까지 다룰 수 있다. 따라서, 유니코드를 구현하기 위해 몇 가지 메커니즘들이 제시되었는데, 가용 저장 공간과, 소스 코드 호환성, 이종 시스템들 간의 상호 교환성들에 기반을 두고 있다.

매핑 방법들은 UTF (Unicode Transformation Format)와 UCS(Universal Character Set) 인코딩이라고 불리며, UTF-32, UCS-4, UTF-16, UCS-2, UTF-8, UTF-EBCDIC과 UTF-7이 포함된다. 뒤에 붙어 있는 숫자들은 UTF 인코딩에서는 하나의 유닛, UCS에서는 한 바이트에 대한 비트 수를 나타낸다. UTF-32 또는 UCS-4에서 한 유닛은 어떤 문자에 대해서도 충분한 용량을 가진다. 다른 경우에는 각 문자를 표현하기 위해서는 유닛의 숫자를 변화시켜야 한다. UTF-8은 내부처리에서 주로 사용되는 UTF-16과 UTF-32 유니코드 텍스트들을 상호 교환하는 인코딩 방식으로, 법적 요건으로 명시된 것은 아니지

만, 사실상의 표준안이다.

유니코드 바이트 순서 표식(Unicode Byte Order Mark : BOM)은 UCS-2와 UTF-16 인코딩에서 텍스트 파일의 첫머리에 명시되는데, 이는 바이트 순서가 필요 없는 UTF-8을 포함한 다른 인코딩을 위해서 소프트웨어 개발자들에 의해 개발되었다. BOM은 U+FEFF가 사용되는데, 어떤 유니코드 인코딩이 사용되느냐에 상관없이 명확한 해석되는 중요한 자질이다. 유닛 FE와 FF는 UTF-8에서는 출현하지 않는다. 또, U+FFFE는 정규 문자가 아니며, U+FEFF는 "넓이가 없는 깨지지 않는 빈칸(Zero-Width No-Break Space)"으로 화면에 표시되지 않는다. UTF-8로 변환된 같은 문자들 또한 바이트 일련체로 EF BB BF가 된다.

2.2.5. 기구성된 문자 vs. 합성 문자 (Ready-made vs. Composite Characters)

유니코드는 문자 형태를 수정하고, 지원되는 그림문자(glyph) 목록을 확장하는 매커니즘을 포함하고 있다. 이것은 구분 기호(diacritic mark)들을 합성하는데 사용된다. 이 구분기호들은 기본 문자 뒤에 삽입되도록 되어 있지만, 호환성 문제로 유니코드에서는 많은 미리 합성된 문자들(precomposed characters)을 제공하고 있다. 많은 경우에 같은 문자를 인코딩하는 여러 가지 방법이 있을 수 있다. 한글도 이와 유사한데, 유니코드는 한글 음절을 한글 자모로 구성하는 방법을 제공하고 있지만, 미리 구성된 한글 음절도 역시 제공하고 있다.

CJK 한자(ideograph)들도 역시 현재는 미리 구성되어 제공되지만, 대부분의 한자들도 더 단순한 기본 요소들로 구성되어 있으며, 원칙적으로는 분해되어 한글처럼 다루어질 수 있어야 한다. 이렇게 되면, 많은 코드 포인트를 줄일 수 있겠지만, 화면에 보여주는 것이 문제가 될 것이다. 또한 Cangjie와 Wubi와 같은

입력에 있어서도 같은 방법을 이용할 수 있을 것이다. 하지만, 문자 인코딩에서 이런 방법을 사용하려는 시도는 한자가 보이는 것처럼 단순히 분해되거나 정규적이지 않다는 문제 때문에 걸림돌이 되어 왔다.

2.2.6. 유니코드를 사용하는 시스템

기술적인 문제와 한계 및 비판에도 불구하고, 유니코드는 주요 인코딩 체계로 부상했다. Microsoft Windows NT와 그를 기반으로 한 Windows 2000 및 Windows XP는 Unicode를 광범위하게 사용하고 있고, 내부 표현으로 UTF-16을 이용하고 있다. Unix계열의 GNU/Linux, BSD, Mac OS X는 유니코드를 받아들이되, 다국어 텍스트를 표현하는 기반으로 UTF-8을 채택하였다.

2.2.7. 입력 방법

Windows XP에서 유니코드 문자를 입력하기 위해서는, Alt 키를 누른 상태에서 유니코드 문자의 십진수 값은 숫자패드로 입력하면 된다. 예를 들어서, Alt 키를 누르고, 960를 입력하면, π (Greek lowercase letter Pi)가 입력된다. 256값보다 작은 값에 대해서는, 코드 페이지 전환을 피하기 위해 0을 먼저 입력하고 입력하면 된다. (예를 들어, Alt 0, 1, 6, 5는 ¥를 표현한다.)

Mac 이용자들도 유니코드의 16진수 값을 직접 입력하는 방법으로 Mac OS X와 Mac OS 8.5 등에서 유니코드를 입력하고 있다. Option 키를 누른 상태에서 유니코드 16진수 4개를 누른다.

2.3. 유니코드와 관련된 이슈

2.3.1. 한글 관련

유니코드에 있어 한글 유니코드는 조합형으로 활용 가능한 한글자모 240자와 완성형 방식으로 사용가능한 한글 11,172자로 구성되었다. 우리의 글을 제대로 표현하기 위해서, 옛한글은 물론 표현가능한 문자는 모두 처리할 수 있어야 하고 중요한 정보처리를 위해서는 음절을 넘어서 자소단위로 처리할 수 있어야 한다. 앞서 언급한 내용처럼, 유니코드는 한 문자를 표현하기 위해, 둘 이상의 방식이 가능하도록 되어 있다. 한글도 마찬가지여서, 조합형과 완성형을 이론적으로는 지원하도록 하고 있다. 하지만, 현재 시스템에서는 완성형을 이용하는 것이 대세이다.

주리정(2001)은 이러한 기본원칙 아래 유니코드에 있어서 한글코드의 문제점을 다음과 같이 언급했다.

첫째, 한글 자모 영역에 현대 한글 자모와 옛한글 자모의 배열이 분리되었다는 점이다. 유니코드는 이 영역 안에서 자모를 초/중/종성으로 나누어 배열하였다. 각각의 초/중/종성 안에서는 현대 한글 자모들을 가나다순으로 먼저 배열한 다음 다시 옛한글에만 쓰이는 자모를 가나다순으로 배열했다. 이러한 배열순서는 정보처리를 용이하게 한다는 기본원칙에도 위배됨은 물론 정렬에 있어서도 문제가 된다.

둘째, 한글 자모 영역의 자모 선정에 기준이 없다는 점이다. 옛 문헌에 보이는 문자 중에서 현대 한글 11,172자에 포함되지 않는 것은 무조건 한글자모에 포함시키는 등 한글 자모를 선정하는 방법이 비체계적이고 비과학적이며 빠진 자모수도 상당히 있다.(홍윤표 1995)

셋째, 옛한글의 경우, 부호화하는 방법만 규정되어 있고 출력장치를 위한 부호

점이 없으며(은광희 1998), 중성과 종성의 조합으로 만들어지는 불완전음절의 경우에는 아예 코드로 지정하지 않고 있다.
옛한글과 관련하여 또 다른 문제점은 새로운 옛한글이 발견되어 코드영역에 추가할 경우 기존의 자모 코드값을 수정하여 추가하거나 코드가 배정되지 않은 기존 자모코드 뒤에 추가해야 한다. 그러나 코드값의 수정 없이 기존 자모코드의 뒤에 새로운 코드를 추가할 경우에는 정렬에 있어서 문제가 된다.

넷째, 구결문자의 경우, 유니코드에 포함시켜야 하면서도 새로운 구결문자가 계속 발견되고 있다는 이유로 포함시키지 않고 있다. 현재 발견된 구결문자는 280여 개 정도이며 한자에서 모양을 빌려왔기 때문에 다른 문자(한자, 그림문자, 기호문자)를 변형시켜 표현하고 있다. 물론 변형을 통해 어느 정도 구결문자의 표현이 가능하지만 완전한 표현을 위해서 구결문자도 유니코드에 포함시켜야 한다.

2.3.2. 한자 관련

주리정(2001)은 한자와 관련하여 다음과 같은 두 가지 문제점을 지적하였다.
첫째, 기존의 표준코드인 KS C 5601:1987과 비교하여 생각했을 때, 한자의 정렬 문제가 발생한다. 유니코드 이전의 표준코드인 KS C 5601:1987의 경우 한자를 한자 음순으로 배열하고, 동음자내에서는 부수별 획수 순으로 배열하고 있어 정렬을 할 경우에도 우리가 흔히 사용하는 한자 음순으로 이루어졌다. 그러나 유니코드 한자는 부수 순으로 배열되어 있어 정렬에 문제가 생긴다. 예를 들어, 유니코드로 작성된 刻(각)과 街(가)의 경우, 刻(각)보다 街(가)가 선행되어 정렬되지 않고 2획의 刂(도)가 부수인 刻(각)이 6획의 行(행)이 부수인 街(가)보다 선행하여 정렬된다.

둘째, 두 가지 이상의 음을 가지는 한자의 배열문제를 언급하였는데, 예를 들

어, 車(차)의 경우 車(거)라고 읽을 수도 있다. 이러한 경우, 기존의 KS C 5601:1987 완성형에서는 상이한 음 모두를 코드 체계에 수용하였다. 이는 한자의 정렬과 해당 한자를 발음에 따라 한글로 변환하는 데에 있어서 모호성을 제거하기 위함이었다.

그러나, 이러한 구분은 우리나라에서만 의미 있는 것으로 유니코드의 경우, 한중일 통합 한자 영역에 상이한 음 중 대표적인 음인 車(차)만을 수용하였고, 車(거)는 기존 KS C 5601:1987 표준과의 변환을 위해 배정된 유니코드의 호환 한자 영역에 배치하였다. 따라서, 두 가지 이상의 음을 가지는 동형이음한자의 경우, 음의 차이를 구별하기 위해 별도로 처리대책을 마련해야 한다.

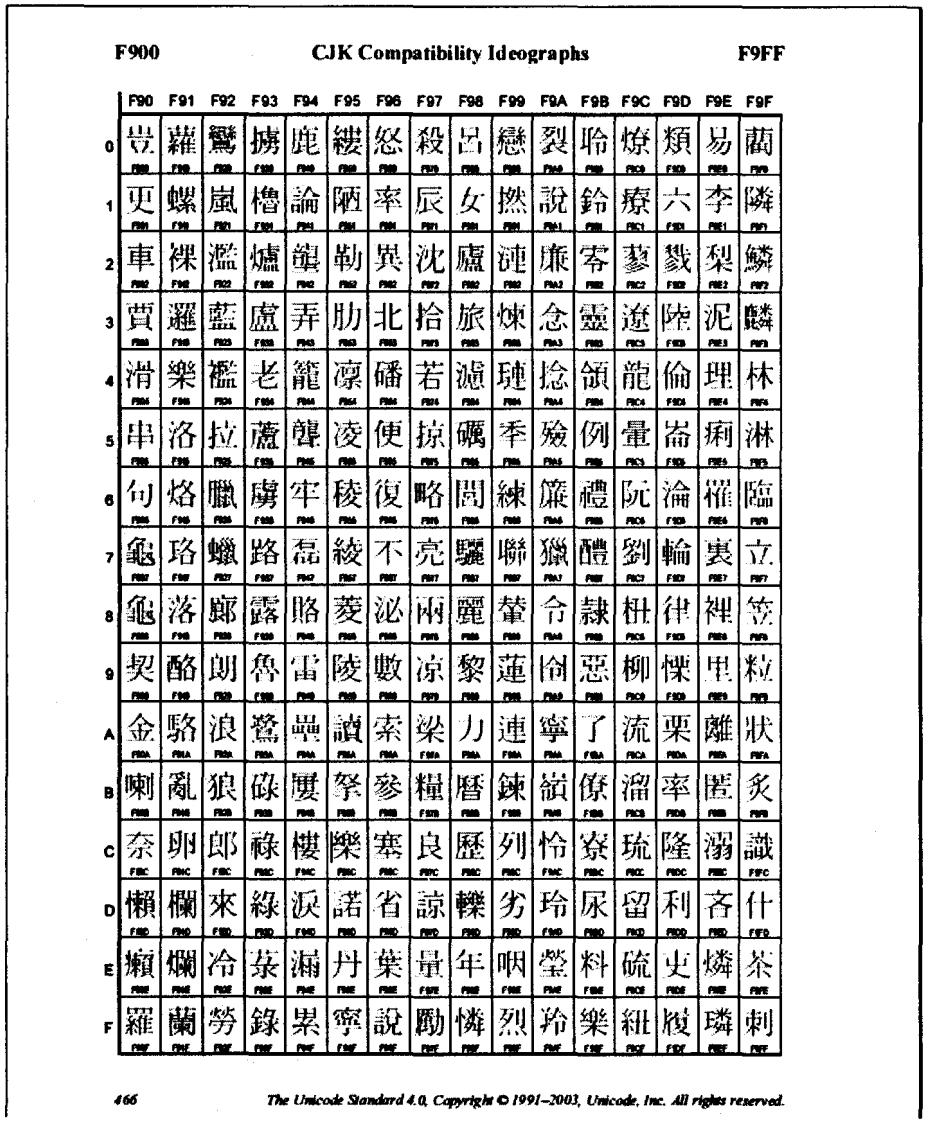


그림 2. 유니코드의 CJK Ideograph 호환영역 - 1

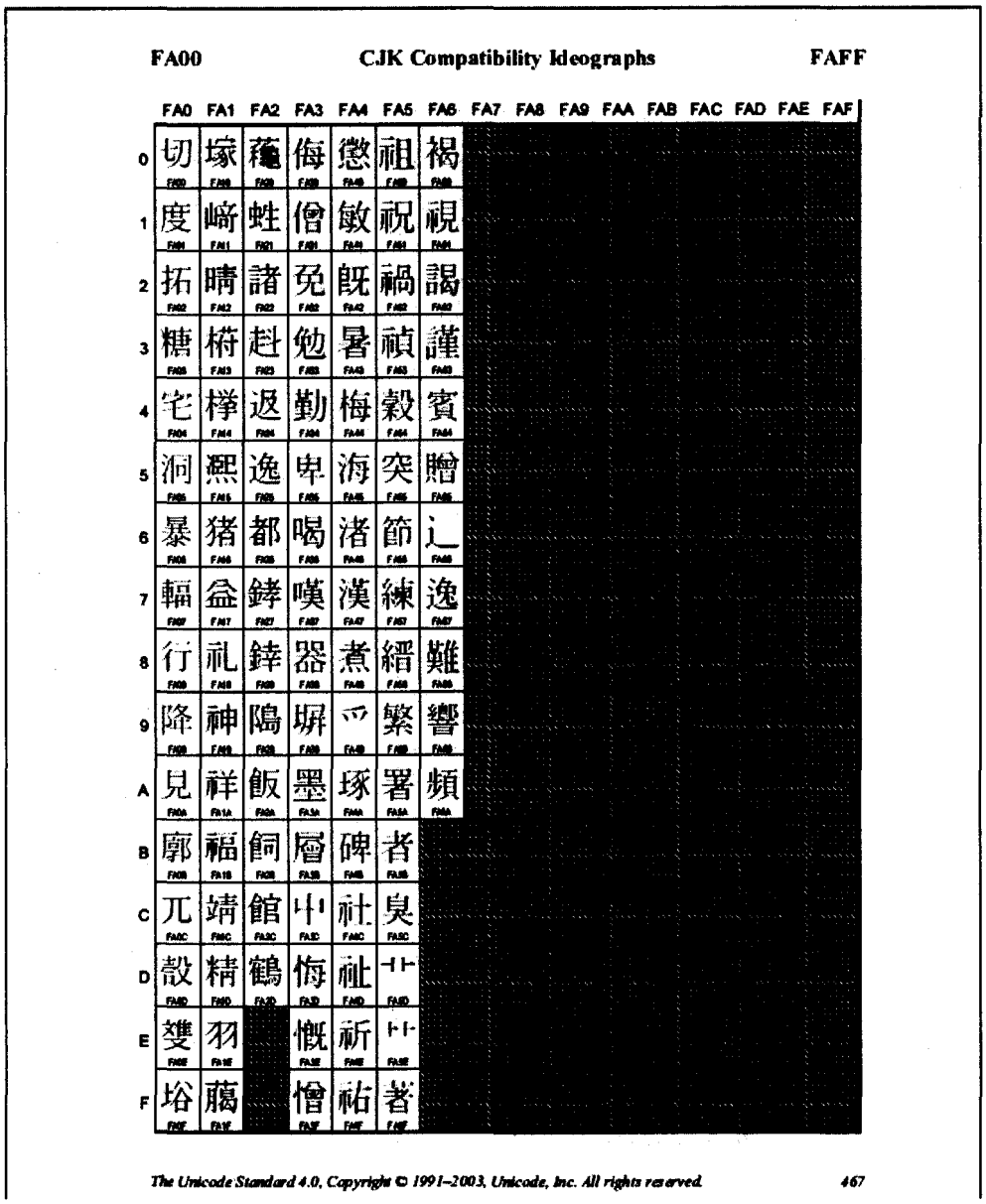


그림 3. 유니코드의 CJK Ideograph 호환영역 - 2

3. 유니코드 시스템 구축

유니코드 구축을 위해서는 기본적으로 서버 OS 및 클라이언트 PC의 운영체제가 유니코드를 지원하여야 한다. 또한, 유니코드를 지원하는 KORMARC 형식이 제정되어야 한다. 이와 같은 기본 인프라를 구축하면, 데이터와 프로그램을 변환하는 작업을 수행하면 된다. 데이터 변환은 DBMS 데이터 변환과 검색엔진 데이터 재생성이 있고, 프로그램 변환에는 서버 쪽에서 운영되는 미들웨어 프로그램이나 클라이언트 쪽에서 운영되는 어플리케이션 프로그램을 모두 변환하여야 한다. 다음은 일반적인 유니코드 시스템 구축 절차를 도식화 한 것이다. 이는 크게 기본 인프라 구축과 실제 전환 절차, 전환 후 관리 등으로 나누어 볼 수 있다.

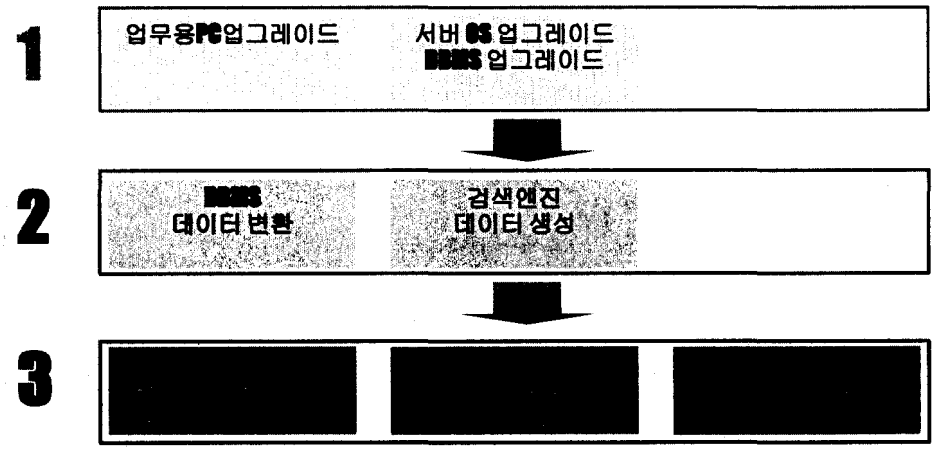


그림 4. 유니코드 시스템 체계 구축 절차

3.1. 기본 인프라 구축

3.1.1. 업무용 PC 업그레이드 및 클라이언트 OS 업그레이드

KS C 5601:1987 체계에서 유니코드를 지원하기 위해서는 서버와 클라이언트 및 PC 등의 환경이 유니코드 지원 환경으로 전환하여야 한다. 이때 웹기반의

대다수의 클라이언트와 PC 등은 별 문제가 되지 않는다. 웹 기반은 서버의 지원이 이루지고, 클라이언트 OS가 유니코드를 지원하면, 클라이언트의 별다른 프로그램 설치 없이 유니코드가 지원될 수 있다. 이를 제대로 지원하기 위해서는 클라이언트와 PC의 업그레이드가 필수이다. 유니코드를 지원하기 위해서는 클라이언트의 운영체제를 Windows XP로 업그레이드 하거나 새로 설치를 하여야 한다. 이때 개인의 사양에 따라 Windows XP로의 전환(Migration) 여부를 알아볼 수 있다. 아래는 Microsoft사에서 권고하는 Windows XP 사양이다.

		내 용
Windows XP Home Edition	참고사이트	http://www.microsoft.com/korea/windowsxp/home/howtobuy/upgrading/sysreqs.asp
	필요 사양	• 300 MHz 이상의 프로세서 속도 권장; 최소 *233 MHz; Intel Pentium/Celeron 제품군, AMD K6/Athlon/Duron 제품군, 또는 호환 가능 프로세서 권장 • 128 MB 이상의 RAM 권장 (최소 64 MB 지원; 단 성능과 일부 기능에 제한이 갈수 있음) • 1.5 GB의 사용 가능한 하드 디스크 공간* • Super VGA (800 × 600) 이상의 해상도를 갖춘 비디오 어댑터와 모니터 • CD-ROM 또는 DVD 드라이브 • 키보드와 Microsoft 마우스 또는 호환 가능한 포인팅 장치
Windows XP Professional	참고사이트	http://www.microsoft.com/korea/windowsxp/pro/evaluation/sysreqs.asp
	필요 사양	• 300 MHz 속도 이상의 프로세서를 가진 PC 권장; 최소 233 MHz 필요 (단일 또는 이중 프로세서 시스템); *Intel Pentium/Celeron 제품군 또는 AMD K6/Athlon/Duron 제품군, 또는 호환 가능 프로세서 권장 • 128 메가바이트 (MB) 이상의 RAM 권장 (최소한 64 MB가 지원되어야 하며, 이 경우 일부 기능의 사용에 제약을 받을 수 있음) • 1.5 기가바이트 (GB)의 가용 하드 디스크 공간* • Super VGA (800 × 600) 이상의 해상도를 갖춘 비디오 어댑터

		와 모니터 • CD-ROM 또는 DVD 드라이브 • 키보드와 Microsoft 마우스 또는 호환 가능한 포인팅 장치

표 5. Microsoft XP를 설치하기 위한 PC 권고 사양

클라이언트 및 PC의 운영체제를 업그레이드 한 후, XP내에 있는 Global IME(Input Method Editor)를 다운 받아 설치하여, 다국어 및 한자 입력이 편리하도록 하여야 한다. IME는 동아시아(한국, 중국, 일본) 여러 나라의 다양한 문자들을 특수 키보드를 사용하지 않고도 입력이 가능하도록 해 주는 프로그램이다.

3.1.2. 서버 OS 업그레이드

유니코드를 지원하는 주요 운영체제의 목록은 다음과 같다. 유니코드 시스템으로 전환하기 위해서는 서버 OS를 Solaris 2.8, 8과 HP-UX 11i 등과 같은 최신 버전으로 업그레이드 해야 한다. 왜냐하면, 최신의 운영체제를 도입하여야 최근의 유니코드 버전을 지원 할 수 있기 때문이다. 다음은 제작사와 유니코드 지원 OS의 목록이다.

제작사	유니코드 지원 OS
Apple	Mac OS 9.2, Mac OS X 10.1, Mac OS XServer, ATSUI
Bell	labs Plan 9
Compaq	Tru64 UNIX, Open VMS
GNU	Linux with glibc 2.2.2 or newer
IBM	AIX, AS/400, OS/2
Microsoft	Windows CE, Windows NT, Windows 2000, Windows XP
SCO	UnixWare 7.1.0
SUN	Solaris 2.6(Solaris 8권장)
HP	HP-UX 11.0(HP-UX 11i권장)
etc...	Inferno by Vita Nuova Java OS Symbian Platform

표 6. 유니코드 지원 운영체제 목록

3.2. 실제 전환 절차

3.2.1. 데이터베이스의 유니코드 전환

유니코드 기반의 데이터베이스를 생성하는 방법은 각각의 DBMS(DataBase Management System)에 따라서 다르다. 그러나, 대부분의 상용 데이터베이스는 Character Set 혹은 Locale 설정, 언어 등의 값에 대한 설정만을 변경하여 주면, 유니코드를 지원할 수 있도록 설계되어 있다. 따라서, 실제 전환(Migration) 단계를 중심으로, 전환 전 고려 사항, 실제 스키마 전환, 전환 후 고려 사항을 포함한 3단계의 전환 단계를 생각할 수 있다. 이 3단계에서 전환 전 고려사항을 엔코딩 방식 선택과 전환할 칼럼 식별로 세분화해서 5단계의 절차를 생각해 볼 수 있다.

일반적으로 KS C 5601:1987 문자셋을 사용하는 DBMS를 유니코드를 지원하는 DB로 전환하기 위해서는 다음과 같은 5단계의 절차를 거친다. 아래는 DBMS를 유니코드를 지원하는 DB로 전환하기 위한 5단계의 세부사항을 기술하고 있다. 거의 모든 DBMS의 유니코드 전환에 대한 일반적인 절차라 할 수 있겠다.

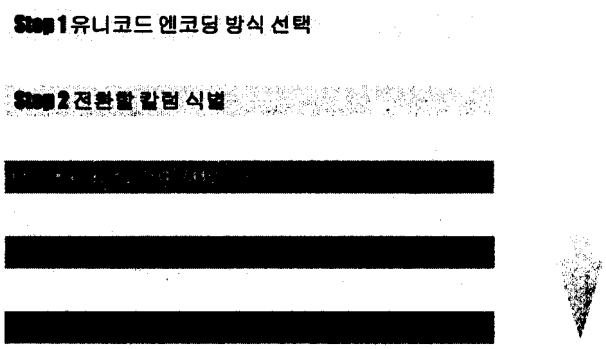


그림 5. DBMS의 유니코드 전환 절차

첫째, 유니코드의 엔코딩 방식 선택(Choosing the Unicode Character Set)을 위한 기준을 마련해야 한다. 유니코드 데이터는 UTF-16과 UTF-8로 인코딩 될 수 있다. 다음과 같은 선택 기준을 바탕으로, 적합한 엔코딩 방식을 결정 할 수 있다.

- 공간 효율성(Space Efficiency)
  영어와 유럽권 언어들은 UTF-8로 인코딩하면 1바이트를 점유하기 때문에 효율적인 반면, 아시아권 언어는 3바이트를 점유한다. 따라서, 아시아권 언어는 UTF-16으로 인코딩하면 모두 2바이트로 인코딩하는 것이 효율적이라고 볼 수 있다.
- 스트링 처리 속도(String processing speed)
  UTF-16이 일반적으로 UTF-8보다 스트링 처리 속도가 빠르다. 따라서 UTF-16이 처리 속도에서 우위에 있다고 볼 수 있다.
- 확장 문자 지원(Supplementary Character support)
  오라클 9i에서 UTF-8은 확장 문자셋(Plane1, Plane2)을 지원하지 않는다. 대신 UTF-16만을 지원 한다.
- 자바(Java)와 윈도우 어플리케이션
  자바와 윈도우 환경은 기본적으로 UTF-16으로 인코딩 되어 있기 때문에 UTF-16을 사용하면 별도의 컨버젼은 필요하지 않다.

둘째, DBMS에서 유니코드 데이터 자체는 전면 혹은 부분적인 전환 모두를 허용한다. 제한된 몇 개의 컬럼만 유니코드를 사용할 경우 부분적인 유니코드 전환만으로도 충분하다고 볼 수 있다. 그러나, 전면적인 전환이 필요할 경우는 시스템 스키마, 시스템 영역을 제외한 모든 영역이 NCHAR 포맷으로 변화됨을 나타낸다.

셋째, DBMS를 전환하기 위해 일반적으로 고려해야 할 사항들을 체크해야 한다. 아래와 같은 일반적인 사항은 대부분의 DBMS전환(Migration)에서 고려되어야 하는 사항이다. 유니코드를 위한 전환에서 실제 고려되어야 할 사항은 다음과 같다.

- **Constraints**

  Primary key와 같은 constraints는 전환을 수행하기 전에 disable시켜야 한다. 즉, constraint를 disable하거나, drop 시킨 후 수행을 하여야 한다. 동시에 두 테이블 이상을 참조하는 constraints가 있을 경우 각각의 테이블을 동시에 전환하여야 한다.

- **Column의 최대 허용 크기**

  CHAR/NCHAR는 2000 바이트 VARCHAR2/NVARCHAR2는 4000 바이트가 최대 허용 가능한 크기이다. 기존 1바이트로 표현되던 문자가 UTF-16인 경우 보통 2바이트, UTF-8인 경우 1바이트에서 3바이트까지 표현된다. 따라서, column의 최대 크기를 넘어서는 경우가 생기지 않도록 주의 하여야 한다. 최대 크기가 넘어서는 경우 NCHAR/NVARCHAR2 대신에 NCLOB 데이터 타입의 사용을 고려해 보아야 한다.

- **Triggers**

  유니코드 전환과정에서 전환되는 테이블과 관련 있는 Triggers는 전환 전에 disable되거나 drop 되어야 할 필요가 있다.

- **Partition**

  저장 구조를 변경시켜야 하는 파티션된 column을 유니코드로 전환하는 것은 매우 복잡한 일이다. 따라서, 전환할 테이블을

export한 후 유니코드 데이터 타입으로 칼럼을 변경한 후 import 하는 방법을 추천한다.

넷째, DBMS 어플리케이션을 유니코드로 전환하기 전에 스키마 전환을 하여야 한다. 다음과 같은 2가지 전환법이 존재한다.(Oracle 데이터베이스의 예)

- **Alter table SQL**

우선, 'alter table' SQL 명령 이용하는 방법으로, alter table SQL 명령을 이용하여 기존의 CHAR나 VARCHAR2를 NCHAR/NVARCHAR2로 변경하거나 NCHAR/NVARCHAR2 칼럼을 테이블에 추가하는 것이다. 예를 들어 다음과 같은 스키마가 있다고 가정하여 보자.

```
student( student_number NUMBER(4), name VARCHAR2(10), major VARCHAR2 )
```

위와 같은 테이블에서 학교 필드를 추가한다고 하였을 때 사용하는 SQL문은 다음과 같다.

```
ALTER TABLE student MODIFY(school NVARCHAR2(20));
```

이때, 해당 컬럼에 인덱스가 잡혀 있을 경우, 데이터 변환에 많은 시간이 소모된다. 따라서, 매우 많은 데이터를 가진 테이블의 경우, 인덱스를 drop 시켜서 없애버린 후 ALTER TABLE SQL을 수행하고, 다시 인덱스를 잡아주는 방법을 추천한다. 그리고, 위와 같은 방법은 CHAR와 VARCHAR2 타입에는 적용이 가능하지만, CLOB 같은 타입에는 적용할 수 없다.

● Online table redefinition

'ALTER TABLE' SQL에 의한 데이터 변환이 장시간 소요되는 경우가 많아, 이러한 경우 서비스 중단의 원인이 된다. 이러한 문제를 해결하기 위하여, DBMS_REDEFINITION 프로시저의 활용을 생각해 볼 수 있다. 위에서 언급한 'alter table' 방법과 Online table redefinition 각각의 특징과 장, 단점은 다음과 같다.

	Alter table	Online table Redefinition
특징	사용하기 쉽고 제약조건이 적다.	많은 양의 데이터를 가진 칼럼에 빠르다.
	Column Definition뿐 아니라 Data들도 NCHAR로 Convert 한다.	한 번에 여러 칼럼을 migration할 수 있다.
	NCHAR, NVARCHAR2일 때 Column length가 2000, 4000 bytes이므로 Migration시 이 SIZE를 넘게 되면 NCLOB 데이터 타입으로의 변경도 생각을 해본다.	Table fragmentation을 피할 수 있다. (space를 절약할 수 있고, data access가 더 빠르다.)
		Migration하는 도중 DML작업을 할 수 있다.
		clob datatype을 nclob datatype으로 바꾸는데 쓰일 수 있다.

표 7. 데이터베이스 테이블 변환을 위한 두 명령어간의 비교

다섯째, 변환을 마친 후에는 변환 전에 변경해 놓았던 항목에 대하여 주의 깊게 원래의 상태로 회복을 시켜 주어야 한다. 변환 전에 고려되었던 항목은 다음과 같다.

항목	내용
인덱스 (Index)	데이터 전환 전에 Drop한 인덱스는 반드시 재생성 해 주어야 한다.
제약 (Constraints)	전환을 위해서 disable 시킨 constraints는 enable 시킨다.
트리거 (Trigger)	전환을 위해서 disable 시킨 trigger를 enable 시킨다.
중복 (Replication)	전환된 칼럼이 여러 DBMS 시스템에 중복된 Replication 테이터라면, 이에 따른 동기화 작업을 수행하여야 한다.

표 8. 데이터베이스의 변환 전 고려항목

위의 5단계 작업 외에도 데이터의 질적 향상을 위해서 '데이터 고품질화' 작업을 생각해 볼 수 있다. 고품질화 작업은 데이터의 Sort와 관련된 것(Data 자체 및 Index 관련)과 움라우트 문자 입력 문제 등을 생각 해 볼 수 있다.

3.2.2. 검색엔진

검색엔진 데이터는 검색 속도 향상을 위하여, DB에서 추출한 자료들을 사용하기 때문에 현재 운용중인 KS C 5601:1987 기반의 검색엔진 데이터를 유니코드 체계로 변화시키는 것이 아니라, 유니코드로 변환된 DB로부터 새로이 데이터를 생성하는 것이다. 이는 신규로 모든 데이터를 생성함으로써 보다 최적화된 검색엔진 데이터 구성이 가능하기 때문이다.

아래는 유니코드 지원 버전과 엔코딩 방식에 따른 검색엔진의 제품별 특징이다.

제품명	유니코드 지원 버전	지원 엔코딩 방식	KS C 5601:1987에 없는 문자
Verity K2	Unicode 3.0	UTF-8	색인 구축 가능
다센 21	Unicode 2.0	UTF-8	색인 구축 가능
Konan DOCRUZE	Unicode 3.0 (2005.2월부터)	UTF-8	색인 구축 가능
Search Formula-1	Unicode 3.0	UTF-8	색인 구축 가능

표 9. 검색엔진의 제품별 특징

다국어로 입력된 자료를 검색시스템에서 검색하고자 하는 경우 시스템은 이용자로 하여금 유사한 영어자모를 쉽게 검색할 수 있도록 해주어야 한다. 즉 독일어의 경우 'U'가 있고 움라우트가 있는 'Ü'가 있을 수 있다. 이 때 실제 목록자료는 'Ü'로 되어 있는 경우 이용자는 'U'를 입력하여 검색을 실행하더라도 해당 목록이 검색 될 수 있도록 처리하는 기능을 지원해야 한다. 이렇게 해야 하는 이유는 움라우트를 글자판에서 직접 입력하지 못하고 문자표 등을 이용해야 하는 경우가 대부분이기 때문이다. 따라서, 시스템은 'Ü'와 'U'등의 모든 경우를 처리하는 방향으로 개발하여야 한다. 위와 같이 다국어에 대한 색인 작성 규칙을 다음과 같이 보다 명확하게 하여야 한다.

Unicode에서 지원되는 다국어는 80여개에 이른다. 그러나, 현실적으로 우리가 시스템을 개발하는데 있어 모든 언어를 다 고려할 수 없다. 따라서, 대략 5~7개 언어를 목표로 개발을 하는 것이 필요하다. 시스템에서 지원한다는 것은 문자의 입력과는 무관하며 입력된 문자를 가지고 색인어를 만들고 검색어를 처리하는데만 영향을 미친다. 즉, 입력된 글자가 문자나 숫자이면 그대로 색인어를 만들고 기호나 특수문자, 공백이면 삭제한다. 또한 라틴계열 언어중 'éàùç'등과 같은 문자는 'eauc' 등으로 바꿔서 색인어를 생성한다.

입력 작업 시 기본적으로 OS에서 지원되는 입력기를 이용하여 모든 다국어의 문자는 입력할 수 있다. 키보드에서 바로 입력할 수 없는 확장문자나 부호, 기호, 특수 문자 등은 '문자표 입력기능'을 지원하여 입력 작업을 수행하도록 한다.

색인어 생성 시 색인어를 만들고자 하는 문자열에 공백이나 기호, 부호, 특수문자 등이 포함되어 있으면 이는 제거하고 만든다. 글자에 악상떼기나 움라우트

등의 기호가 결합된 문자는 유사한 문자로 변경하여 색인을 생성한다. 검색 시, 검색어로 입력된 문자열은 위에서 언급한 동일한 방법을 따른다.

3.3. 전환 후 관리

유니코드 기반 시스템과 KS C 5601:1987 기반 시스템 사이에는 많은 데이터 교환이 근본적으로 발생한다. 이러한 데이터의 교환을 자연스럽게 지원하기 위해서는 국립중앙도서관을 유니코드 사용기관으로 보았을 때 외부 기관들에 따라 아래와 같이 하면 된다.

반입 및 반출유형 / 파일 유형	외부기관의 반입 시	외부기관으로 반출 시
유니코드 파일 사용기관	별도의 변환 없이 바로 반입	별도의 변환 없이 바로 반출
KS C 5601:1987 파일 사용기관	해당 파일을 유니코드로 자동 변환 후 반입	해당 파일을 유니코드로 자동 변환 (데이터 손실 발생할 수 있음)

표 10. 외부기관별 반입/반출을 위한 파일 형식

그 밖에 데이터 고품질화를 위한 데이터 및 인덱스 정렬과 움라우트 입력에 따른 문제는 오라클 데이터베이스에서의 다국어 전환에서 다루어져 있으므로 참조를 바란다.

3.4. 오라클과 데이터베이스에서 다국어 전환 (세부사항 포함)

3.4.1. Multilingual Support Database의 기본 Architecture

다국어로 된 서비스와 자원을 제공하고자 할 경우, 개별 언어별로 시스템을 구축하는 것은 구축과 관리 측면에서의 비용 증가가 Business의 효율성을 저하시킬 것이다. 여기에 대한 대안이 I18N(Internationalization), L10N (Localization)에 기초한 Multilingual Support Database를 구축하는 것이다. 아래는 Multilingual Support Database에서의 Client/Server의 개요를 나타낸 구성도

이다.

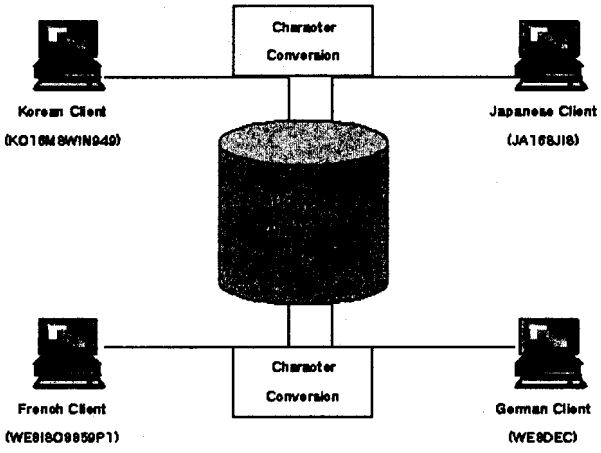


그림 6. Client/Server의 다국어지원 데이터베이스

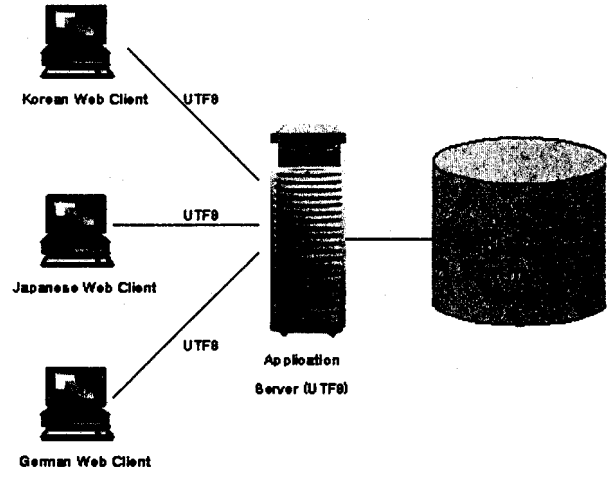


그림 7. 다계층환경의 다국어지원 데이터베이스

Globalization/Multilingual Support Database의 실제 구축 작업에 있어서는 DB뿐만 아니라 시스템 작업 환경에 대해 세부적으로 고려해야 할 사항이 매우

많아진다. 아래는 세부적으로 고려되어야할 실제적인 요소들에 대한 내용이다.

3.4.2. 유니코드(Unicode)와 오라클의 유니코드 지원

대표적으로 사용되는 유니코드로는 UTF-8, UCS-2, UTF-16이 있으며 각각의 특징과 장단점을 도표로 나타내면 다음과 같다.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
UTF-8	8 bit	가변 길이 - 유럽어 : 1~2 - 아시아어 : 3(대부분), 보충(supplementary) character : 4 byte	ASCII	가변 길이므로 유럽어 저장시 공간을 덜 차지한다. ASCII의 Superset이므로 ASCII에서 마이그레이션이 쉽다.
UCS-2	16 bit	고정 길이 - 유럽어 : 2 byte - 아시아어 : 2 byte, 보충(supplementary) character : 지원 안함		아시아어 저장시 공간을 덜 차지한다. Java나 Microsoft 클라이언트와 호환성이 좋다.
UTF-16	16 bit	고정 길이 - 유럽어 : 2 - 아시아어 : 2 보충(supplementary) character : 4 byte	UCS-2	아시아어를 저장시 공간을 덜 차지한다. Java나 Microsoft 클라이언트와 호환성이 좋다.

표 11. 유니코드 전송방식별 특징

가변 길이는 언어마다 한 글자가 차지하는 길이(byte)가 다르다는 것을 말하고, 고정 길이는 언어에 관계없이 한 글자가 차지하는 길이는 동일하다는 것을 의미한다. 각 유니코드별 장점도 이러한 특성을 고려하면 쉽게 이해 할 수 있을 것이다.

오라클의 경우 V7에서는 유니코드 버전 1.1을 지원하는 AL24UTFFSS - V7

에서만 사용가능 - 도입하면서 유니코드를 지원하기 시작하였고 V8에서 유니코드 버전 2.0을 지원하는 UTF-8이 도입되고 V9에서는 유니코드 버전 3.1을 지원하는 AL32UTF-8 / AL16UTF-16이 도입되어 유니코드를 지원하고 있다. 자세한 것은 아래 표를 보면 알 수 있다.

Character set	지원 버전	유니코드 인코딩	유니코드 버전	Database character set	National character set
AL24UTFFSS	7.2-8i	UTF-8	1.1	지원	지원 안함
UTF-8	8.0-9i	UTF-8	8.0-8.1.6 : 2.1 8.1.7이후 : 3.0	지원	9i만 지원
UTFE	8.0-9i	UTF-8	8.0-8.1.6 : 2.1 8.1.7이후 : 3.0	지원	지원 안함
AL32UTF-8	9i	UFT-8	9iR1 : 3.0 9iR2 : 3.1	지원	지원 안함
Al16UTF-16	9i	UFT-16	9iR1 : 3.0 9iR2 : 3.1	지원 안함	지원

표 12. Oracle의 버전별 유니코드 지원 현황

3.4.3. 다국어 지원을 위한 데이터베이스 구축 방안

지금까지 유니코드와 오라클에서 지원하는 유니코드에 대해서 살펴보았다. 다국어를 지원하는 데이터베이스를 생성하기 위해서 오라클에서 지원하는 두 가지 방법이 있는데 하나는 유니코드를 데이터베이스 character set으로 지정해서 유니코드 데이터베이스를 구축하는 방법이고, 다른 하나는 데이터베이스 character set에 관계없이 NCHAR (National Character) 데이터타입을 이용하는 방법이다. 오라클 9i부터 NCHAR 데이터타입에 유니코드 character set을 지정할 수 있게 되었으므로 두 번째 방법은 9i부터 고려할 수 있는 방법이다.

비교적 방법 자체는 간단하다고 할 수 있다. 데이터베이스 생성 시 데이터베

이스 character set에 유니코드를 지정해서 char나 varchar2에 다국어를 저장하거나 데이터베이스의 character set을 변경하지 않고 다국어를 저장할 칼럼을 NCHAR (National Character) 데이터타입으로 설정해서 다국어를 저장하면 된다. 그렇지만 방법을 결정함에 있어서 프로그래밍 환경, 마이그레이션의 용이성, 성능, 데이터 타입, 어플리케이션의 타입 등 많은 요소들을 고려해야 하기 때문에 결코 쉬운 일은 아니다.

이제부터 두 방법을 사용할 때 어떤 character set을 지정하는 것이 유리한가를 살펴볼 것이다. 유니코드 데이터베이스 구축하면서 지정 가능한 character set은 UTF-8, AL32UTF-8, UTFE이다. UTF-8과 AL32UTF-8은 유니코드에서 지정한 UTF-8과 거의 동일 특성을 가지고 있는데 UTF-8의 경우는 보충(supplementary) character가 6byte를 차지한다는 점이 다르다. UTFE는 EBDIC platform을 위한 것이므로 일단 논외로 하겠다. 남은 UTF-8과 AL32UTF-8 중에서 UTF-8과 동일한 특성을 지닌 AL32UTF-8을 사용할 것을 권장한다. AL32UTF-8은 보충(supplementary) character가 4byte를 차지하므로 UTF-8에 비해서 공간을 덜 차지하고 유니코드에서 정한 UTF-8과도 동일하기 - 유니코드3.1기준에 맞기 - 때문에 보충(supplementary) character를 처리하면서 data conversion의 부하가 발생하지 않기 때문이다.

NCHAR 데이터타입의 character set은 데이터베이스 생성 시 NATIONAL CHARACTER SET에 지정된 값에 따르며 UTF-8과 AL16UTF-16을 지정할 수 있다. UTF-8은 가변 길이기 때문에 스토리지를 절약할 수 있지만 처리 시 스페이스를 붙이는 부하가 생기며, AL16UTF-16은 고정 길이이기 때문에 처리 시 스페이스를 붙이는 부하는 없지만 스토리지의 낭비를 초래할 수도 있다. 따라서 저장한 언어가 주로 어느 나라 언어인지 성능과 스토리지 절약 중 어떤 것에 중점을 둘 것인가를 고려해서 결정해야 한다.

여러 고려 요소들을 떠나서 가장 이상적인 방법은 AL32UTF-8을 database character set으로 정하고 UTF-8을 지원하는 언어로 프로그램을 개발해서 character set conversion이 없이 데이터가 오가는 것이다.

3.4.4. 스키마 설계 시 고려해야 할 점과 데이터 타입

테이블 설계 시 고려할 점으로는, 다국어를 저장하고자 할 경우 언어별로 칼럼을 달리해서 저장하는 방법, 특정 칼럼에 모두 저장하는 방법을 생각할 수 있다. 업무의 특성과 여러 요인을 고려하여 방법을 선택해야 겠지만 가장 이상적인 방법은 동일 칼럼에 모두 저장하는 방식이다. 그렇지만 이 경우 칼럼에 있는 데이터만으로는 어느 언어인지 알 수 없으므로 언어정보를 관리할 칼럼을 추가해서 설계해야 한다. 이러한 언어정보 칼럼의 추가는 다음 연재에서 다루어질 Linguistic sort와도 밀접한 관계를 가지며 특정 언어로 된 데이터만을 검색하고자 하는 요구를 만족시킬 수 있는 유일한 방법이다.

데이터 타입을 정할 때 고려해야 하는 요소는 다음과 같다. 데이터 타입과 길이를 정할 때는 database character set에 가변 길이인 UTF-8 / AL32UTF-8만 지정할 수 있다는 한계 점을 인식하고 접근하는 것이 좋다. V9 R2에서부터 문자데이터 타입의 경우 데이터의 길이를 기술함에 있어서 BYTE 또는 CHAR이라는 키워드를 사용하여 BYTE 단위 또는 문자 단위로 기술 할 수가 있다. NCHAR 데이터 타입의 경우는 문자단위로 밖에 길이를 기술할 수 없다. 사용상의 주의 점은 데이터 타입과 길이 기술 방식 모두에 관계가 있으므로 각 길이 기술 방식에 대해서 알아보자. 간단한 사용 예를 보이면 다음과 같다.

예) CREATE TABLE lang_test

(ename1 CHAR(4), ename2 CHAR(4 CHAR),
ename3 VARCHAR2(4), ename4 VARCHAR2(4 CHAR),
ename5 NCHAR(4),
ename7 NVARCHAR2(4))

그렇다면 발생할 수 있는 경우의 수들에 대해서 길이를 4byte 또는 문자 4개로 주고 'a엔' 이라는 데이터를 insert한 경우 length()와 lengthb()의 결과가 어떻게 나타나는지에 대해서 살펴보자.

데이터타입								
기술방식	byte	char	byte	char	char		Char	
Character set	UTF-8 / AL32UTF-8		UTF-8 / AL32UTF-8		UTF-8	AL16UTF-16	UTF-8	AL16UTF-16
길이	4	4	4	4	4	4	4	4
입력데이터	'a엔'	'a엔'	'a엔'	'a엔'	'a엔'	'a엔'	'a엔'	'a엔'
저장데이터	'a엔'	'a엔bb'	'a엔'	'a엔'	'a엔bb'	'a엔bb'	'a엔'	'a엔'
Length	2	4	2	2	4	4	4	2
Lengthb	4	6	4	4	6	8	4	4
테이블에 만들어지는 컬럼 길이	4	12-UTF-8 16-AL32UTF-8	4	12-UTF-8 16-AL32UTF-8	12	8	12	8

표 13. Oracle 자료형별 특징 (* b는 공백을 뜻함.)

여러 결과를 종합해 보면 다음과 같은 결론을 내릴 수 있다. CHAR 데이터타입의 경우 우리가 잘 알고 있듯이 길이의 선언 방식에 관계없이 기술된 길이보다 작으면 정해진 길이가 될 때까지 채운다. 이때 채우는 길이가 다른데 byte의 경우는 부족한 byte 만큼 스페이스를 채우고 문자 개수의 경우는 부족한 문자 개수만큼 스페이스를 채우며 스페이스가 차지하는 길이는 해당 character

set의 최소 길이다. 좀 더 거시적인 관점에서 얘기한다면 CHAR이므로 정해진 길이보다 모자란 부분은 스페이스로 채운다.

사용상의 주의 점에 대해서 알아보자. VARCHAR2/NVARCHAR2의 경우는 가변 길이이므로 발생할 데이터의 전체 길이에 맞게 길이를 잘 정하는 것 외에는 사용상에 주의 점은 없다. 그러나 고정 길이인 CHAR/NCHAR의 경우는 다르다. 길이를 문자 개수로 선언하거나 데이터 타입을 NCHAR로 선언한 경우 해당 컬럼내에서 언어에 따라서 길이가 달라지므로 값을 비교할 경우 길이가 다른 CHAR 타입을 비교할 때와 마찬가지로 스페이스 패딩이 일어난다. 따라서 CHAR 타입을 사용할 경우는 byte로 길이를 명시하고 NCHAR를 사용하고자 할 경우는 character set을 AL16UTF-16으로 하여 저장 및 비교 시 스페이스 패딩의 부하를 최소화 하여야 한다. 나아가서는 CHAR나 NCHAR 보다는 VARCHAR2, NVARCHAR2를 사용하여 스페이스 패딩 부하와 저장 공간 낭비를 줄이는 것이 좋다.

3.4.5. 클라이언트의 NLS_LANG 설정

다국어 데이터베이스를 구축하기 위해서 서버 쪽에서 어떻게 할 것인가에 대해서 살펴보았다. 그렇다면 클라이언트의 경우를 알아보자. 직접적인 방법을 언급하기에 앞서 NLS_LANG의 역할을 살펴보자. NLS_LANG에 Language, Territory, Characterset을 기술하게 된다. 이 중에서 Language는 오라클의 메시지나 요일이나 월등을 나타내는 언어를 의미하고 Territory는 통화나 숫자를 나타내는 형식, 주(week)를 계산하는 방식을 의미하며 Characterset은 클라이언트의 character set을 의미한다.

오라클은 NLS_LANG에 설정된 Characterset의 값이 클라이언트의 O/S 또는 클라이언트 프로그램의 character set설정에 맞게 설정되어 있는지 확인하지

않는다. 따라서 클라이언트의 character set이 오라클 데이터베이스와 다를 지라도 NLS_LANG이 데이터베이스의 character set과 동일하게 설정되어 있으면 character set conversion을 하지 않고 데이터를 저장하게 된다. 이렇게 되면 손상된 데이터가 저장되게 되므로 Characterset의 설정에 주의해야 한다.

따라서 NLS_LANG에 설정된 Characterset의 값은 클라이언트의 O/S 또는 클라이언트 프로그램의 character set에 맞게 설정을 해 주어야 한다. 그렇다면 클라이언트의 character set은 무엇에 의해서 결정될까? 윈도우의 경우는 ACP (ANSI Code Page)에 의해서 결정되고 Unix의 경우는 터미널의 character set에 의해서 결정된다. 윈도우의 경우는 윈도우 화면에서와 도스 프롬프트에서의 ACP가 다른 경우 - 한글윈도우의 경우는 같다 - 가 있으므로 설정에 주의해야 한다. 윈도우 화면에서의 ACP는 HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Control₩Nls₩CodePage₩ACP 라는 레지스트리 정보를 이용하면 알 수 있고, 도스 프롬프트에서는 chcp라는 명령을 통해서 알 수 있다. 이렇게 하면 코드 값이 나오는데 한글 윈도우에서는 949이고 이는 오라클 character set 중에서 KO16MSWIN949에 해당된다. 값과 오라클 character set의 매핑에 대한 정보는 Globalization support를 참조하면 된다. 유닉스의 경우는 터미널의 환경정보를 이용하면 된다.

NLS_LANG의 Characterset을 설정함에 있어서 반드시 클라이언트의 O/S character set을 따라야 하는 것은 아니다. 클라이언트에 프로그램을 사용하여 데이터를 주고 받을 경우 이 프로그램이 UTF-8을 지원한다면 O/S character set에 관계없이 UTF-8으로 설정도 가능하다.

3.4.6. Sorting의 종류와 설명

3.4.6.1. Binary Sort

Binary Sort는 각 문자를 encoding하기 위해 사용된 binary값에 의해 정렬되는 기법이다. ASCII나 EBCIDIC의 기준에 의해 각각의 A~Z의 문자를 binary값으로 지정하여 그 순서에 따라 정렬하며, UNIX상에서는 ASCII로, 그 밖의 대다수 환경에서는 EBCIDIC방법을 따른다.

인코딩 방식	정렬 순서
ASCII	숫자 -> 대문자 알파벳 -> 소문자 알파벳
EBCIDIC	소문자 알파벳 -> 대문자 알파벳 -> 숫자

표 14. 인코딩별 정렬 순서

수행속도가 가장 빠른 정렬 방법 중의 하나이다. 하지만 해당 언어의 encoding방식이 각 문자에 지정된 binary값의 순서와 같은 경우만 정렬이 올바르게 되는 단점이 있다. 결국 각 언어별로 적합한 정렬 결과를 보장 받기는 힘들다. 영문 알파벳에 가장 적합한 방법으로, NLS_LANGUAGE가 AMERICAN(default)으로 지정된 경우나 binary_sort인 경우에 binary sorting의 방법으로 정렬된다.

3.4.6.2. Linguistic Sorting

각 언어에 적합한 정렬방식을 선택 하는 방법으로 Binary sorting보다 데이터 프로세스 양이 더 많다. Linguistic Sorting에는 Monolingual과 Multilingual의 두 가지 방법이 있다.

3.4.6.2.1. Monolingual Linguistic Sorting

각 언어에 적합한 정렬 순서에 맞추어 문자에 숫자 값을 지정하여 재정의 하는 방법이다. 각각의 숫자 값은 Major값과 Minor값을 저장하고 있는 Table을 참조하여 정렬한다.

각각의 문자는 두 개의 major값과 minor값을 배정 받아 위에서 언급한 Table에 저장된다. 이 값을 이용하여 두 단계에 거쳐 정렬을 하게 된다. 알파벳 정렬이 끝나고 나면 각각의 문자는 major값과 minor값으로 변환이 된다. 그 두 개의 값으로 binary 정렬방식에서처럼 숫자 값의 순서에 따라 데이터를 정렬한다.

유럽 문화권 언어에 맞게 정렬이 가능하다는 장점이 있다. 아시아 언어에는 적절하지 못하다는 단점이 있다. Oracle에서 한국어나 중국어는 monolingual 정렬방법이 제공되지 않는다. 언어별로 Sorting에 관한 정보는 Oracle의 Globalization Support Guide를 참조 할 수 있다. 주로 유럽지역 언어에 적합한 정렬방식으로 알파벳을 기본으로 하는 언어에 많이 사용된다. 또한 DB에 한가지의 언어로 저장된 경우에는 monolingual sorting방식이 더 효율적이다. Multilingual을 사용하는 경우엔 불필요하게 정의된 문자로 인해 메모리 사용이 증가하는 비효율이 발생한다.

3.4.6.2.2. Multilingual Linguistic Sorting

Oracle 9i부터 monolingual 정렬방법의 확장된 정렬방법으로 유럽어 이외의 동양권 언어에도 사용이 가능한 정렬 방법이다. 더 많은 문자가ISO 14651과 Unicode 3.1을 기준으로 정의 되어 있다.

Multilingual sorting작업은 3단계를 거친다. Primary, Secondary, Tertiary를 거쳐 정렬 작업이 수행된다. 동양권 언어인 경우엔 ISO 14561에 기준하여 그

나라의 언어에 맞게 정렬된다.

Primary Sorting	Base character의 구분을 하는 단계로 a와 b가 다른 단어라는 근본적인 구분 단계
Secondary Sorting	Base character + Diacritics까지 구분하는 단계
Tertiary Sorting	Base character + Diacritics + Case까지 구분하는 단계

표 15. 소팅 방식별 특징

가령 monolingual 정렬방식에서 multilingual 정렬 방식으로 정렬방법을 수정한다면 NLS_SORT = French에서 French_M으로 변경하면 된다. 이 경우 Generic_M(알파벳을 기준으로 하는 언어의 정렬)으로의 정렬을 기초로 하고 거기에 추가로 French 정렬을 하게 되는 것이다.

Multibyte character인 동양권 언어나 정렬 기준이 난해한 정렬에 적합하다. 특히 한국어나 중국어 같이 monolingual 정렬로는 불가능했던 언어가 multilingual 정렬로 가능해 졌다. 무엇보다 한 번의 정렬로 여러 개의 언어를 동시에 정렬 할 수 있다는 장점이 있다. 특정언어에 대한 정렬과 동시에 라틴어를 기초로 하는 문자에 대한 정렬을 동시에 할 수 있다. 예를 들어 Korean_M이라고 정렬을 하는 경우 한국어와 라틴어를 기초로 한 문자가 동시에 정렬이 가능하게 된다. 하지만, 중국어나 일본어도 테이블에 포함되어 있다고 한다면 이 언어는 동시에 정렬이 불가능하다. 지정된 정렬 언어도 아니고 라틴어를 기초로 하는 문자도 아니기 때문이다. 한 번의 정열작업으로 한 개 이상의 언어를 정렬이 가능하므로 데이터가 다국어로 저장되어 있는 경우에 multilingual 정렬이 추천되는데 이는 한 가지의 언어만 저장된 경우 multilingual sorting을 사용하여 정렬하면 수행속도가 monolingual을 사용한 경우보다 저하된다. 이 같은 현상은 multilingual sort에 더 많은 문자가 정의 되어 있기 때문이다. (정렬 지정 언어 + Generic_M)

3.4.6.3. 각각의 Sorting 비교

불어를 다음과 같이 입력한 경우 소팅의 종류별로 다른 결과가 나온다.

⇒ Diet, À voir, Freizeit, Đń

Binary Sorting	Monolingual Sorting	Multilingual Sorting
(Select ~Order by name)	(Select ~Order by NLSSORT(name, NLS_SORT=FRENCH))	(Select ~Order by NLSSORT(name, NLS_SORT=FRENCH_M))
Diet	À voir	À voir
Freizeit	Diet	Diet
À voir	Freizeit	Đń
Đń	Đń	Freizeit

표 16. 소팅 방식간의 비교

Binary Sorting의 경우, Diacritic과 'Đ'를 인식하지 못하여 정렬이 되지 못한 경우이고 monolingual sorting의 경우에는 'Đ'가 불어문자 체계에 없는 문자라 인식하지 못하여 정렬작업에 참여를 하지 못했으나 multilingual sorting의 경우엔 참여가 가능하여 모든 문자가 정렬된 경우이다. Multilingual의 경우엔 라틴어를 기초로 하는 언어와 불어가 동시에 정렬되어 다음과 같이 모두 정렬이 가능하다.

3.4.7. Migrating to Oracle9i NCHAR Datatypes

3.4.7.1. Migrating Oracle8 NCHAR Columns to Oracle9i

Oracle9i에서는 NCHAR Datatype의 경우 UTF-8, AL16UTF-16의 Unicode Character set Encoding방식만을 채택한다. 따라서 Oracle8 버전에서 지원이 되었던 "JA16SJISFIXED"와 같은 Asian Character set은 더 이상 지원을 하지 않게 되었다. 8 버전의 NCHAR, NVARCHAR2, NCLOB 컬럼을 9i의 NCHAR Datatype으로 변경하는 방법은 다음과 같다.

1) Export all NCHAR columns from the Oracle8 or Oracle8i database. 2) Drop the NCHAR columns 3) Upgrade database to Oracle9i. 4) Import the NCHAR columns into Oracle9i.

표 17. Oracle에에서 데이터타입을 변경하는 순서

또 다른 방법으로는 NCHAR Upgrade script인 $ORACLE_HOME/rdbms/admin/utlchar.sql을 이용해도 Oracle8의 NCHAR Column을 Oracle9i의 NCHAR Column으로 Upgrade할 수 있다. 이 스크립트를 한 번 실행 한 뒤에는 Downgrade가 불가능하다. Downgrade를 하려면 모든NCHAR Column을 drop을 하고 다시 import를 받아야 한다. 따라서 이 스크립트를 실행하기 전에 export를 받아 놓는 것이 좋다.

3.4.7.2. Changing the National Character Set

앞에서 다국어 통합 Database 생성 시 National Character Set의 선택과 이용방법에 대해서 살펴보았었다. National Character Set을 UTF-8에서 AL16UTF-16으로 또는 그 반대로 변경할 경우 간단하게 Alter database national character set new_character_set; 명령으로 바꿀 수 있다. 이 경우 ALTER DATABASE CHARACTER SET 과 ALTER DATABASE NATIONAL CHARACTER SET 을 함께 적용하는 것이 좋다.

3.4.7.3. Migrating CHAR Columns to NCHAR Columns in an Oracle9i Database

Oracle9i Database에서 CHAR 컬럼을 NCHAR 컬럼으로 변경하는 방법에는 다음 두 가지가 있으며, 각각의 특징들이 있다.

첫째, Alter table table_name modify 명령어를 사용하면, 사용하기 쉽고 제약조건이 적고, Column Definition뿐 아니라 Data들도 NCHAR로 Convert한다. 반면, NCHAR, NVARCHAR2일 때 Column length가 2000, 4000 bytes이므로 Migration시 이 SIZE를 넘게 되면 NCLOB 데이터 타입으로의 변경도 생각을 해 봐야 한다.

둘째, Online table redefinition 명령어를 사용하면, 많은 양의 데이터를 가진 컬럼에 빠르고, 한 번에 여러 컬럼을 migration할 수 있으며, Migration하는 도중 DML작업을 할 수 있다. 또, Table fragmentation을 피할 수 있어서 space를 절약할 수 있고, data access 가 더 빠르다는 장점이 있다. 이외에도 clob datatype을 nclob datatype으로 바꾸는데 쓰일 수 있다. 이 명령어를 사용하기 위한 구체적인 순서는 다음과 같다.

1) dbms_redefinition.can_redef_table('scott', 'emp');

2) empty interim table을 create한다.

```
CREATE TABLE int_emp (   empno number,
                         ename nvarchar2(10),
                         job nvarchar2(9),
                         org nvarchar2(10) )
```

3) online table redefinition

```
dbms_redefinition.start_redef_table  (   'scott',
                                         'emp',
                                         'int_emp',
                                    'empno empno, to_nchar(ename) ename,
                                    to_nchar(job) job, to_nchar(org) org' )
```

※ clob를 nclob 칼럼으로 migration할 경우는 to_nclob를 쓴다.

4) interim table에 triggers, indexes, grants, constraints등을 생성한

다. Referential constraint는 disabled mode로 만든다.

Trigger는 online Table redefinition process가 끝나면 실행이 된다.

5) Original table과 interim table을 sync를 맞춘다.

Original table에 DML작업이 많다면 dbms_redefinition.sync_interim_table procedure를 실행한다.

```
Ex) dbms_redefinition.sync_interim_table('scott', 'emp', 'int_emp');
```

6) dbms_ redefinition.finish_redef_table 프로시저 실행

```
Ex) dbms_redefinition.finish_redef_table('scott', 'emp', 'int_emp');
```

이 프로시저가 실행되면 original table은 interim table의 모든 attributes, indexes, constraints, grants, triggers로 재정의 된다.
Original table의 referential constraints가 적용된다.

7) interim table을 drop 한다.

3.4.8. CHARACTER SET SCANNER

Character Set Scanner는 Character Set Migration을 할 때 Database내에 있는 모든 Data를 확인하고 발생할 수 있는 문제점들을 나타내어 준다. 이 작업은 Character Set Migration시 반드시 해야 할 필수 작업으로서 이후 나오게 되는 Summary Report는 우리가 하는 Migration의 방향을 알려준다고 할 수 있다. Character Set Scanner는 다음과 같이 세 가지 Mode에서 scanning할 수 있다.

1. Full Database Scan
2. User Scan

3. Table Scan

이는 우리가 Migration하는 Data의 범위에 따라 결정이 된다. Character Set Scanner를 실행하는 방법을 정리하면 다음과 같다.

1. Before using character set scanner
2. Csminst.sql을 실행한다. (오직 한 번만 실행) 실행 순서는 다음과 같다.
3. create a user named CSMIG
4. assigns the necessary privileges to CSMIG
5. assigns the default tablespace to CSMIG
6. connects as CSMIG
7. creates character set scanner system tables under CSMIG

※ CSMIG의 default tablespace는 system이므로 Csminst.sql를 편집하여 default tablespace를 바꾸어 준다.

```
%cd $ORACLE_HOME/rdbms/admin
%sqlplus "system/XXXXX as sysdba"
SQL>start csminst.sql
```

3.4.8.1. Invoking the Character Set Scanner

Character Set Scanner를 실행하는 방법은 다음과 같이 세 가지로 나뉜다.

- Using parameter file : csscan system/manager parfile=filename
- Using the command line
  : csscan system/manager full=y tochar=UTF-8 array=10240 process=3

● Using an interactive session

3.4.8.2. Getting Online Help for the Character Set Scanner

앞서 언급했던 대로 Character Set Scanner를 실행하면 Report를 생성하게 된다. 다음과 같이 두 개의 Reports를 생성하게 된다.

3.4.8.2.1. Database Scan Summary Report

Database Scan Summary Report는 다음의 section으로 되어 있다.

- Database Parameters for the Character Set Scanner
- Database Size
- **Scan Summary**
- **Data Dictionary Conversion Summary**
- Application Data Conversion Summary
- **Application Data Conversion Summary for Each Column Size Boundary**
- **Distribution of Convertible Data for Each Table**
- Distribution of Convertible Data for Each Column
- Indexes To Be Rebuilt

위의 Section중에서 Bold체로 표시된 중요한 section들에 대해서 그 내용을 보면 다음과 같다.

● Scan Summary

Database character set migration이 가능한지를 알려준다. 하나는 Data dictionary의 조건, 다른 하나는 Application data의 조건이다.

Possible Data Dictionary Status	Possible Application Data Status
All character-type data in the data dictionary remains the same in the new character set.	All character-type application data remains the same in the new character set.
All character-type data in the data dictionary is convertible to the new character set.	All character-type application data is convertible to the new character set.
Some character-type data in the data dictionary is not convertible to the new character set.	Some character-type application data is not convertible to the new character set.

표 18. Data Dictionary와 Application Data의 상태 조건

위의 표에 나와 있는 형식대로 표현이 된다. "all the data remains the same in the new character set"이라고 나올 경우는 original character set의 encoding은 target character set의 encoding과 같고, alter database character set명령으로 migration할 수 있다. "all the data is convertible to the new character set"이라고 나올 경우는 안전하게 Export & Import utility를 이용하는 것이 좋다. 그러나 이 경우에도 encoding방식이 같을 수도 있고 다를 수도 있다.

● Data Dictionary Conversion Summary

이것은 full database scan이 실행될 때만 나오게 된다. 나오는 형식은 다음과 같다.

Status	Description
Changeless	Number of data cells that remain the same in the new character set.
Convertible	Number of data cells that will be successfully converted to the new character set.
Exceptional	Number of the data cells that cannot be converted. If you choose to convert anyway, some characters will be lost or data will be truncated.

표 19. Data Dictionary의 상태

만약 Convertible과 Exceptional column의 값이 0 이면 모든 데이터가 convert없게 된다. Exceptional column 값이 0 이고, Convertible column값이 zero가 아니면 모든 dictionary에 있는 데이터가 새로운 character set으로 convert되어야 한다. 만약, Exceptional column이 0이 아니라면 dictionary 데이터 중에서 convert되지 않을 데이터가 있다는 의미이다.

● Application Data Conversion summary for Each Column Size Boundary

char와 varchar2 application data의 conversion summary를 나타낸다. 이것은 boundaries parameter를 지정했을 때만 나오게 된다.

Status	Description
Changeless	Number of data cells that remain the same in the new character set.
Convertible	Number of data cells that will be successfully converted to the new character set.
Exceptional	Number of the data cells that cannot be converted. If you choose to convert anyway, some characters will be lost or data will be truncated.

표 20. Application Data 변환 요약

● Distribution of Convertible Data for Each Table

Database내에서 Convertible, Exceptional data가 얼마나 분포되어 있는지 보여준다. 이 자료가 많을수록 Database내에서의 Convertible Data가 많음을 의미한다.

3.4.8.2.2. Individual Exception Report

Individual Exception Report는 다음의 section으로 되어 있다.

1. Database Scan Parameters
2. Data Dictionary Individual Exceptions
3. Application Data Individual Exceptions

● Database Scan Parameters

scan의 type과 parameter를 나타내 준다.

Parameter	Value
Scan type	Full database
Scan CHAR data?	YES
Current database character set	we8mswin1252
New database character set	utf8
Scan NCHAR data?	NO
Array fetch buffer size	102400
Number of rows to heap up for insert	10
Number of processes	1

● Data Dictionary Individual Exceptions

convertible, exceptions를 가지는 Data dictionary를 나타내 준다.

Exceptions에 관해서는 다음의 두 가지 경우가 있다.

1. Exceed column size
2. Lossy conversion

● Application Data Individual Exceptions

Exceptions를 가지는 data를 나타내고 필요시 수정을 해야 한다.

n Exceed column size

Maximum column width를 넘게 되면 column size를 늘려야 한다.

n Lossy conversion

Lossy conversion의 데이터는 migration전에 수정이 되어야 한다. 만약 수정을 하지 않으면 replacement character로 변환된다.

WE8ISO8859P1의 character set을 UTF-8로 변환 시 Individual exception report의 예제는 아래와 같다.

```
User:   HR
Table:  EMPLOYEES
Column: LAST_NAME
Type:   VARCHAR2(10)
Number of Exceptions: 2
Max Post Conversion Data Size: 11

ROWID              Exception Type      Size Cell Data(first 30 bytes)
------------------ ------------------ ----- ------------------------------
AAAA2fAAFAABJwQAAg exceed column size    11 Åhrenfeldt
AAAA2fAAFAABJwQAAu lossy conversion          óråclëø™


AAAA2fAAFAABJwQAAu exceed column size    11 óråclëø™
------------------ ------------------ ----- ------------------------------
```

Migration시 **Åhrenfeldt**와 **óråclê8™**는 column size (10)을 넘게 된다. 왜냐하면 WE8ISO8859P1 상황에서는 **Å. ó. å.**와 **ê** 등의 문자가 1-byte를 차지하게 되지만, UTF-8 상황에서는 2-byte를 차지하기 때문이다. 또한 **óråclê8™**은 lossy conversion이다.

```
UPDATE hr.employees SET last_name = 'Oracle8 TM'
WHERE ROWID = 'AAAA2fAAFAABJwQAAu'
```

Trademark sign은 WE8ISO8859P1 의 유효한 데이터가 아니기 때문이다. 따라서 이런 경우는 다음과 같은 데이터 조작을 통해 변경해 줘야한다.

4. 국내외 유니코드 지원 사례 분석

4.1. 국내 사례

유니코드를 지원하는 도서관 시스템을 개발하였다고 관련 업계에서 홍보를 하고 있으나, 실제적으로 대학도서관 수준의 정보 서비스 기관에서 유니코드 시스템으로 전환을 한 사례는 찾아보기 어렵다. 더욱이 완전한 유니코드 체계로 전환한 사례는 더욱 드물다 하겠다.

도서관 시스템이 완전한 유니코드 체계를 지원하기 위해서는 소장 자료의 정보가 유니코드로 저장이 가능하고, 유니코드로 입력, 편집, 검색이 가능하여야 한다. 또한, 목록, 상호대차, 원문 서비스 등 각종 도서관 서비스가 유기적으로 연동되어야 한다. 그리고, 외부 도서관과의 정보교환을 위해서는 유니코드 MARC 반입, 반출을 지원해야 한다. 그러나, 아직 유니코드 문자셋을 지원하는 KORMARC가 발표되지 않았다. 따라서, 국내에 완전한 유니코드 체계를 가진 도서관 시스템이 존재하지 않는 이유는 당연한 것일 수 있다. 국립중앙도서관, 한국외국어대학교, 광주과학기술원, 국어연구원, 한국 역사 정보 통합시스템 등의 국내와 외국의 유니코드 서비스 시스템 구축 사례를 살펴보겠다.

4.1.1. 국립중앙도서관 (http://www.nl.go.kr)

4.1.1.1. 개요

2003년 5월에 정보시스템에 대한 유니코드 체계로의 전환을 위한 정보화 전략 계획(ISP)에 착수하여 유니코드 체계로의 전환에 따른 문제점과 해결방안을 모색하고 있는 상태이다. 그리고, 유니코드 문자세트를 수용하는 KORMARC에 대한 초안을 2003년 중에 발표하였다. 2004년 12월 현재는 조달입찰로 '목록, 목차'만 유니코드로 변환하였고, DB를 P/G로 일괄 유니코드로 변환하였다. 2005년 1월에 유니코드 시스템을 위한 전용 홈페이지를 개통[1]하였다.

1) http://unicode.nl.go.kr

4.1.1.2. 단점

KS C 5601:1987에 정의되지 않은 다국어 문자, 한글 고어의 지원은 불가능하다. 2003년 현재, 유니코드 데이터 반입 및 국가 간 서지 데이터의 상호 교환 불가능한 상태이다.

4.1.2. 한국외국어대학교 (hhttp://weblib.hufs.ac.kr/)

4.1.2.1. 개요

한국외국어대학교 도서관은 학교의 특성상 KS C 5601:1987로 표현할 수 없는 다국어 자료들이 타 대학교에 비해서 많은 편이다. 따라서, 이들 자료의 목록 작성 및 검색 서비스를 위하여 유니코드 시스템 도입을 서두른 것으로 생각된다.

4.1.2.2. 특징

윈도우2000 이상의 운영체제 환경에서 DBMS로 MS-SQL 서버를 이용하고 있으며, 문자세트로는 UTF-8을 사용하고 있다. 유니코드의 입력은 운영체제에서 지원하는 IME를 이용하여 웹을 통한 검색을 수행하고 그 결과를 볼 수 있다. 현재 독어, 불어와 같은 로마자 형태 자료는 일부 음독기호(ex. @, <, >, *, ~, #, ^)를 이용하여 입력한다

KS C 5601:1987에서 지원하지 않는 한자는 한글로 입력하고 KORMARC 890태그의 $h를 활용하고 있다. 기존의 KS C 5601:1987에서 지원하지는 않지만 불완전하나마 입력과 검색이 가능한 자료는 기존의 도서관 시스템에서 관리하고, 비로마자 형태의 자료는 별도의 유니코드 시스템으로 관리하고 있다.

비로마자 형태의 외국어(베트남어, 아랍어, 이란어, 태국어, 힌디어) 자료는 10,000여건 정도 보유하고 있으며, 이들 자료에 대한 서지정보가 신규 유니코드 시스템에 구축되어 웹을 통해 검색이 가능하다.

4.1.2.3. 단점

기존의 도서관 시스템과 연동되지 않고 독립적으로 존재하는 형태로 구축되어 있어 기존 MARC 데이터베이스와 통합관리가 불가능하다. 또한, MARC 레코드를 이용한 서지 정보 교환도 불가능하다.

4.1.3. 광주과학기술원 (http://www.kjist.ac.kr/)

4.1.3.1. 개요

광주과학 기술원은 국내에 수입되고 있는 방대한 양의 외국학술지를 공동 활용할 목적으로 1998년부터 2000년까지 과학기술부와 정보통신부에서 추진한 과학기술정보화근로사업을 통해 외국학술지 목록구축 작업을 수행하였다. 이렇게 구축된 목록을 토대로 외국학술지의 분담수서 및 공동 활용 체계의 지속적인 발전을 위하여 '학술지 공동 활용협의회' KORSA(Korea Resource Sharing Alliance)를 결성하였으며, 현재 120여개 도서관에서 소장하고 있는 24,000여 종의 외국학술지에 대한 목차 및 소장 정보를 구축하여 서비스하고 있다.

4.1.3.2. 특징

현재 KORSA 시스템의 운영체제는 윈도우2000, DBMS는 미국 InterSystems 사의 Cache를 채용하고 있으며 유니코드를 지원하고 있다. 데이터베이스 스키마 전환 시, Cache가 각 필드의 데이터 길이를 명시하지 않기 때문에 비교적 손쉽게 이루어 졌다. 또한, 색인을 추출하고 관리하는 별도의 검색엔진을 사용하지 않고 DBMS 자체의 검색 기능을 사용했기 때문에 큰 문제가 되지 않았다.

4.1.4. 국어 연구원 (http://www.korean.go.kr)

4.1.4.1. 개요

국어연구원은 1999년부터 유니코드 작업을 시작하였다. ms-word2000을 사용하여서 유니코드 데이터를 구축하였다. ms-word2000의 경우는 실제 유니코드 테이블을 사용한 것이 아니고 어플리케이션에서 코드를 확장하여 KS C 5601:1987에서 지원되지 않는 한자 또는 옛글자를 지원하였다. 따라서 추후에 유니코드로의 변환작업이 반드시 있어야 한다. ms-word2002(Windows는 2000이상)로 구축한데이타는 실제 유니코드 체계로 구축된 것이므로 추후에 유니코드로의 변환은 필요 없다.

4.1.4.2. 특징

한글97 조합형으로 데이터를 구축하였다. 내부적으로는 유니코드 기반이라 볼 수 없으므로 추후에 유니코드 표준이 확정되면 데이터에 대한 변환 필수적이다. 다만, 홈페이지 서비스 시 유니코드로 변환하여 보여주고 있는데, 홈페이지에서 확장한자나 옛글자를 검색하거나 입력하기 위해서는 MS사에서 지원하는 유니코드 폰트와 입력기를 다운받아야 한다.

국립국어연구원의 검색 시스템은, 유니코드를 사용하므로 다음 사항이 충족되어야 검색 기능을 제대로 활용할 수 있다.

가. 운영체계 : Windows 98/NT

나. 브라우저 : Internet Explorer 5.0 이상 사용 (내려받기)

다. 옛글자 글꼴(옛글자 및 확장 한자)

- MS Word 2000에서 지원하는 옛글자 및 확장 한자 글꼴 사용

라. 옛글자 입력기

- 옛글자 입력기 및 입력창 ActiveX 모듈 사용

옛글자 및 확장한자, 옛글자 입력기 등은 MS사에서 지원하는 기능을 다운받아서 사용함

4.1.5. 한국 역사 정보 통합시스템 (http://www.koreanhistory.or.kr)

4.1.5.1. 개요

2000년부터 유니코드 사업을 시작하였다. 데이터는 구축 시부터 유니코드 체계를 따랐으므로 데이터 변환은 없었다.

4.1.5.2. 특징

한글2002를 이용하여 데이터를 구축하고 있으며, 유니코드 폰트나 한적입력기를 자체 개발하였지만, 유니코드 표준을 따르고 있지 않다.

유니코드 기반으로 구축되어 있다고 하나 현재 제정된 유니코드 표준 형태를 따른 것이 아니므로 국어연구원과 마찬가지 경우로 유니코드의 표준이 제정된 후 데이터 변환은 필수[2]이다.

4.2. 국외 사례

4.2.1. 일본 NII 사례 (http://www.nii.ac.jp/index-j.html)

4.2.1.1. 개요

일본 국립정보학연구소(NII, National Institute of Informatics)는 정보학 관련 종합 연구 및 학술정보 유통을 위한 기반 기술 개발 및 정비를 목적으로 2000년 4월에 설립된 기관이다.(1986년 4월에 설립된 문부성 산하의 학술정보센터

2) MS Word 2000, 워디안, 한글2002의 경우 유니코드 체계를 따르는 것이 아니고 사용자 정의 영역을 확장하여 KS C 5601:1987에서 지원하지 않는 한자를 지원하는 것으로 유니코드로의 변환은 필수이다. 그러나, Windows2000운영체제에서 MS Word 2002로 DB를 구축하는 경우는 유니코드 형태로 데이터가 구축되므로 추후에 유니코드로 변환하지 않아도 된다.

를 폐지 전환하여 설립된 기관임) NII는 목록정보 서비스, 정보검색 서비스, 전자도서관 서비스, 학술정보 네트워크 등의 학술정보 서비스를 제공하고 있다. NII 목록 시스템의 전환과정 및 다국어 대응에 대하여 살펴보기로 한다.

4.2.1.2. NII 목록 시스템

NII의 목록 시스템(NACSIS-CAT)은 1984년 12월에 운용을 시작하였다. 그 후, 하드웨어의 소형화와 고속화를 통한 다운사이징 바람이 일어났으며, 소프트웨어 측면에서는 오픈 시스템, C/S 시스템(client/server system), GUI(graphic user interface) 조류가 일반적인 것이 되었다. 이에 따라 구 NACSIS-CAT과 ILL 시스템도 변화하는 적응하는데 한계가 있다고 판단하였다. 그래서, 1995년 신(新) CAT-ILL 시스템 검토회의를 설치하고, 신규 시스템이 갖추어야 할 주요한 내용에 대하여 논하였다. 그 내용은 다음과 같다.

4.2.1.3. [신(新) CAT-ILL 시스템 검토회의] 주요 논의 내용

첫째, 다운사이징과 인터넷 시대에 맞는 새롭고 유연한 시스템이 필요하다.

둘째, 급격한 이용자 증가에 따른 고가용성, 고안정성, 고성능 시스템이 필요하다.

셋째, 다국어 문자에의 대응이 필요하다.

4.2.1.4. NII 목록 시스템의 다국어 대응

1997년에 [신(新) CAT-ILL 시스템 검토회의]에서 제시된 요구사항에서 신 CAT 시스템(C/S 시스템)을 개발하였으나 다국어 대응은 구현되지 않았었다.(왜냐하면, 그 당시에는 UCS 처리 환경이 정비되어 있지 못하였다.) 따라서, 종합목록 데이터베이스에는 중국어, 한국어 등의 자료에 대한 등록은 전체적으로 진행되지 못하였다. 전국의 대학 도서관에는 약 700만권의 중국어 자료, 약 70만권의 한국어 자료를 소장하고 있었다. 그러나, 목록 시스템이 다국어를 지원하

지 못하였기 때문에, 정보의 유통과 교육, 연구에 큰 지장을 초래하였다.
2000년 1월 전산기 교체와 맞추어서, 데이터베이스에 수록되어 있는 모든 데이터를 UCS로 변환하였다. 그리고, 신CAT 시스템이 다국어(중국어, 한국어...)를 지원할 수 있도록 함으로써 각 도서관 소장 자료의 목록 작성이 가능해졌다.

4.2.2. 일본 NII의 신CAT 시스템

4.2.2.1. 개요

다국어 대응의 신CAT 시스템은 다국어 데이터를 보존하는 데이터베이스의 변경과 다국어 데이터를 입출력하는 서버의 변경으로 구현된다. 아래는 신CAT 시스템의 변경사항을 나타낸 것이다.

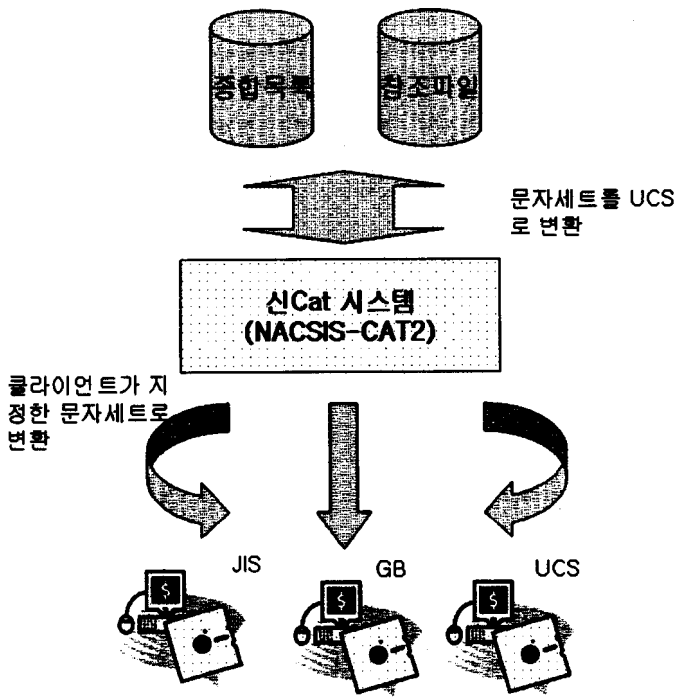


그림 8. 신CAT 시스템의 개요

4.2.2.2. 신 CAT 시스템의 다국어 대응

목록 소재정보 데이터베이스(종합목록 데이터베이스, 참조파일)의 문자세트를 일본어 EUC에서 UCS로 변경한다. 신CAT 시스템의 서버에서 클라이언트로 데이터를 보낼 때 문자세트로 JIS와 GB, UCS도 추가한다. 도서관 시스템 측에서는 중국어 전용으로는 GB 대응 클라이언트, 중국어 이외의 문자도 취급하려면 UCS 대응 클라이언트가 필요하다.

4.2.2.2.1. 데이터베이스의 다국어 대응

데이터베이스 문자세트를 UCS로 변경하는 것으로써 수록 데이터는 다음 표와 같이 표현된다. 기존의 EXC문자와 UCS 외자(클라이언트 문자코드로 변환할 수 없는 UCS 문자를 말한다. 형식은 ◆U0x◆ 이다. 이때 0x는 UCS16진 코드를 의미한다.)는 UCS로 정의되고 있는 확장 알파벳이나 한자로 변환된다. 중국어 간체자(간체자)와 번체자, 한글 등 대부분의 문자가 UCS로 정의되어 있으므로 표시가 가능해졌다.

변경전	변경후
영숫자(JIS X0201), 가나, 한자(JIS X0208)	대응하는 영숫자, 대응하는 일본어 (가나, 한자)
발음식별부호 첨부문자(EXC 문자)	대응하는 확장 라틴 문자
한자, 간체자 등의 화자	대응하는 한자

표 21. 데이터 베이스 수록 데이터

4.2.2.2.2. 신CAT서버의 다국어 대응

기존의 서버에서는 JIS7 인코딩만 대응할 수 있었지만, 신규 시스템에서는 GB, GBK, UTF-8, ISO 2022 JP의 4종류의 인코딩에 대응할 수 있게 되었다.

각 인코딩 방식을 요약하면 아래와 같다.

ISO 2022	EXC 문자를 포함하지 않는 JIS7 문자세트를 취급하는 것이어서 ECC 문자는 UCS 외자로 표시된다.
GB	간체자 중심이 중국어용 문자세트로 일본어 표시는 할 수 없다.
GBK	GB의 확장판으로 UCS와 동등의 CJK 통합한자를 포함한다.
UTF-8	UCS의 문자세트에 대한 인코딩 방식

표 22. 신CAT 서버의 인코딩 방식 비교

지정한 문자셋트에 의한 데이터 표현은 아래 표와 같다. O표시는 해당하는 문자로 표현할 수 있는 것이고 X표시는 ◆U0x◆와 같이, 1문자씩 UCS코드를 나타내는 번호로 표현할 수 있음을 의미[3]한다.

영숫자(JIS X0201)	O	O	O	O	O
가나, 한자(JIS X0208)	O	O	△*	△*	O
발음식별 부호 첨부문자(EXC문자)	O	X	△**	△**	O
중국어(GB 2312)	△***	△***	O	O	O
UCS 밖에 없는 한자 간체자 등	X	X	X	O	O
한글	X	X	X	X	O

표 23. 신 CAT 서버의 인코딩 방식들의 UCS 데이터 기술능력 비교

3) * 히라가나, 가타가나 및 중국어에 대응하는 것이 있는 한자만 해당하는 문자로 표현 가능
** 한자를 중국어 발음으로 표시하는데 사용되는 음표부호 첨부 문자는 해당하는 문자로 표현 가능
*** 일본어에 대응하는 것이 있는 한자만 해당하는 문자로 표현 가능

4.2.2.2.3. 클라이언트의 다국어 대응

UCS에 의한 데이터의 표시와 입력을 실시하기 위해서는, UCS에 대응하는 CATP 클라이언트가 필요하다. 구체적으로는 윈도우 2000등과 같이 UCS에 대응할 수 있는 CATP 클라이언트가 필요하다. 구체적으로는 윈도우 2000 등과 같이 UCS에 대응할 수 있는 OS에서 동작하는 UCS문자세트의 입력 틀을 갖춘 클라이언트를 개발할 필요가 있다. CATP 클라이언트를 개발하고 있는 각 도서관 시스템 업체들로 하여금 UCS 대응 클라이언트의 개발을 진행하도록 하고 있다. 또한 NII에서 개발한 UCS 대응 클라이언트 WebUIP를 2001년 1월부터 공개하였다. WebUIP는 웹에 대응한 신CAT/ILL 시스템 게이트웨이 시스템으로 표준적인 웹브라우저를 CATP 클라이언트로 사용하여 시스템으로 표준적인 웹브라우저를 CATP 클라이언트로 사용하여 CATP 서버를 이용할 수 있다. WebUIP 자체는 CATP 서버의 외부에 위치해 웹브라우저로부터 서비스 요청을 받아 CATP 서버를 경유하여 데이터베이스를 이용하는 구성을 취하고 있다. 인터넷 익스플로러5 등의 UCS(UTF-8)의 입출력이 가능한 브라우저 환경에서 중국어, 한국어 자료 등의 입력과 디스플레이가 가능해 졌다.

4.2.3. 미국 OCLC 사례 (http://www.oclc.org/)

4.2.3.1. 개요

1971년부터 OCLC서비스의 기반이 되고 있는 종합목록 WorldCat은 운용을 시작하여, 현재 9,000여 회원 기관이 공동으로 목록 작성에 참여하고 있으며, 현재 40여개 언어로 구현된 5,000만 건 이상의 서지레코드를 보유하고 있는 세계적인 서지 데이터베이스이다. OCLC는 종합 목록 이용기관들의 다양한 요구를 충족시키기 위하여 자동 편목(PromptCat) 서비스, 갱신목록 통지(Bibliographic Record Notification) 서비스, 정부문서목록(GovDoc) 서비스, 맞춤형 편목(Custom Cataloging) 서비스, 일괄 처리(Batch Processing) 서비

스 등 각 기관의 특성에 적합한 목록 서비스를 지원하고 있다.

4.2.3.2. OCLC의 목록 시스템의 다국어 대응

4.2.3.2.1. 개요

OCLC의 목록 서비스는 WorldCat을 근간으로 하여 제공되는 서비스로 OCLC 목록 서비스의 핵심은 WorldCat을 유지/관리하는 종합목록 시스템이라고 볼 수 있다. 30여 년 전에 운용을 시작한 종합목록 서비스 시스템은 정보 기술 및 환경의 변화에 대응하여 지속적인 성능 개선과 추가기능 개발을 통해 발전되어 왔으며, 21세기에 걸맞은 차세대의 WorldCat을 구축하기 위하여 전면적인 시스템 전환을 서두르고 있다. 2002년 7월에 OCLC에서 발간한 뉴스레터에 의하면 WorldCat 데이터베이스 플랫폼을 기존의 독자적인 데이터베이스 관리 시스템(DBMS)에서 오라클 DBMS로 전환하는 계획을 수립하였으며, 전환된 시스템에서는 MARC 뿐만 아니라 더블린 코어(Doublin Core), TEI, EAD와 같은 여러 메타데이터 형식의 지원은 물론 유니코드를 지원함으로써 다국어에 대응할 수 있도록 하고 있다. 이 계획에 따르면 오라클 플랫폼에서 서비스되는 차세대 WorldCat은 2003년 말에 완성될 예정이다.

4.2.3.2.2. 특징

WorldCat과 Resource Catalog는 ALA 문자세트만이 지원된다. ALA 문자세트는 영어권 국가의 도서관에서 널리 사용되는 문자세트로 로마자 알파벳, 특수문자, 그리고 로마자에 기반을 둔 공통적으로 이용되는 발음식별기호를 지원한다. ALA문자세트로 구성된 OCLC MARC 또는 UTF-8 형식의 DC를 반출한다. Connexion 전거파일도 ALA문자세트만을 지원하며 반출은 ALA 문자세트로 구성된 MARC21 형식을 취한다.

Pathfinder는 완전한 유니코드를 지원하며, Dewey 서비스는 유니코드 디스플

레이를 지원한다. Connexion은 전 세계의 사용자를 위한 웹 기반 시스템으로의 잠재력을 확보하기 위하여 웹 브라우저와의 통신 방식과 내부 저장 형식으로 UTF-8을 사용한다. 아래의 표는 Connexion에서 지원하는 문자세트를 정리한 것이다.

Conexion Db	반입	편집/검증	생성	반출
WorldCat and Resource Catalog	유니코드 변환	ALA문자세트에 대응되는 유니코드	ALA문자세트	OCLC MARC:ALA 문자세트 DC HTML& DC RDF: UTF-8
Connecxion Authority File	N/A	ALA문자세트에 대응되는 유니코드	ALA문자세트	MARC21:ALA문자세트
Pathfinder DB	유니코드 변환	유니코드	유니코드	UTF-8
WebDewey	N/A	N/A	N/A	N/A

표 24. Connexion에서 지원하는 문자세트

WorldCat은 ALA문자세트로 표현할 수 없는 자료 중 상당부분을 차지하고 있는 한국어, 중국어, 일본어, 페르시아어, 터키어 등의 자료를 입력할 수 있도록 CJK 편목 소프트웨어와 Arabic 편목 소프트웨어를 제공하고 있다. Arabic 편목 소프트웨어는 아라비아어 문자로 작성된 자료에 대한 편목을 지원하는 윈도우 기반의 프로그램이다.(CJK 문자를 디스플레이하기 위해서는 윈도우NT, 윈도우2000, 윈도우XP와 같은 유니코드를 지원할 수 있는 OS를 갖추어야 한다.)

5. 서울대학교 도서관 유니코드 전환 시 고려사항

5.1. 서울대학교 도서관 유니코드 적용의 필요성

5.1.1. 국가통합서지 관리기관의 유니코드 체계 검토

앞에서 살펴본 것처럼, 국외 기관들의 유니코드 체계 검토에 발맞추어, 국립중앙도서관에서는 "국립중앙도서관 정보시스템 유니코드 체계 구축 정보화전략 보고서"(2003.10)를 출간하면서 2004년 국립중앙도서관의 일부 시스템 혹은 특정 시스템에 유니코드 체계를 도입 단계를 거쳐 2005년에 현재 시범 운영하고 있으며[4], 2006년에 전면 구축 등으로 점진적으로 구축하려고 계획 중이다. 또한 한국교육학술정보원(KERIS)에서는 "학술 데이터베이스의 유니코드 변환 적용에 관한 연구"(2003년)를 출간하는 등, 여러 기관들이 유니코드 체계 전환을 검토하고 있다.

5.1.2. 다국어 자료에 대한 고품질 데이터베이스 구축

서울대학교 도서관은 전문 학술 자료 및 외국 자료를 다량 보유하고 있으며, 이를 위한 고품질 서지 데이터베이스를 작성 및 관리해야 할 필요가 있다. 서울대 도서관에서 보유하고 있는 외국어 자료의 분포 현황을 다음과 같다.

언어구분	건수(비율)	언어구분	건수(비율)
영어	445,381(39.48)	스웨덴어	108(0.01)
한국어	353,736(31.36)	덴마크어	97(0.01)
일본어	136,231(12.08)	포르투갈어	79(0.01)
독일어	76,770(6.81)	폴란드어	65(0.01)
중국어	37,708(3.34)	투르크멘어	52(0.00)
프랑스어	28,556(2.53)	아랍어	49(0.00)
스페인어	5,315(0.47)	아제르바이잔어	48(0.00)

4) http://unicode.nl.go.kr/index.php

러시아어	4,873(0.43)	팔리어	42(0.00)
이탈리아어	2,194(0.19)	체코어	38(0.00)
라틴어	1,194(0.11)	인도네시아어	30(0.00)
터키어	880(0.08)	노르웨이어	26(0.00)
헝가리어	272(0.02)	키르기스어	23(0.00)
네덜란드어	240(0.02)	몽골어	22(0.00)
그리스어(고대)	145(0.01)	핀란드어	21(0.00)
그리스어(현대)	67(0.01)	루마니아어	19(0.00)

표 25. 서울대 도서관의 외국어 자료 분포 현황 (2004년 4월 기준)

이런 다국어 자료들을 KS C 5601:1987에서 모두 표현할 수 없으므로, 서울대학교 도서관에서는 임의의 기호 규칙을 만들어 다국어 데이터를 구축하였다.

첫째, 890 태그를 사용하여 한글이나 다른 문자로 대체하여 작성하였다. KS C 5601:1987에서는 한자를 4,888자만을 지원하므로 한자가 없어서 한글로 입력하는 경우가 있었는데, 그 통계는 다음과 같다.

변형문자표시	내용	건수	비고
890 $b	한글 고어나 특수문자를 입력	1,160	
890 $h	한자가 없어서 한글로 입력	21,413	
890 $r	로마문자로 변형하여 입력	9	
890 $x	한자 이외의 외국문자를 한글로 입력	200	
계		22,782	

표 26. 한글 및 다른 문자로 대체한 경우

둘째, 존재하지 않는 문자를 특수 문자나 기호를 통해 대체 입력하거나, 셋째, 로마나이즈로 입력하는 방법을 사용하였는데, 그 방법의 예는 아래 표와 같다.

언어구분	특수문자 입력방법	비고
프랑스어	'(악쌍떼귀) → > '(악쌍그라브) → < ç(세디유) → * "(트레마) → @ ^ (악쌍시르꽁플레스) → 무시함	
독일어	"(움라우트) → @ β(에스체트) → ss	
스페인어	'(악센또) → > ñ(띨드) → * "(디에르시스) → @	
중국어	"(특별발음 부호) → @ ? (한글움이 없는 경우) → ?? / 로마나이즈	
일본어	¯(장음부호) → ^	

표 27. 특수 문자 입력의 예

5.1.3. 국외 기관과의 데이터베이스 교류 활성화

미국의 국회도서관(http://www.loc.gov/)은 유니코드 문자세트를 수용하는 MARC21[5]을 제안하였다. 이에 의하면, 2002년도 MARC21의 서지, 소장, 전거 레코드를 유니코드로 구축하기로 계획하였으며, 2003년 1월, 유니코드로 1차 테스트 변환하여, 변환된 목록[6]에 대한 검증을 수행하였다. 또한, 일본의 NII는 이미 유니코드로 자료를 변환하였다.

이와 같이 유니코드로 저장되어 있는 자료들을 현재 KS C 5601:1987에서 받아들일 경우, 일부 문자들이 소실되거나 변형된 형태로 저장하여 서비스해야 하

5) http://www.loc.gov/marc/

6) 1,200만 서지, 500만 전거, 2,000만 소장레코드

는 불편이 있다. 따라서, 국외 기관과 다양한 도서관 자원의 유통을 위해 유니코드 수용이 필요하며, 고품질 데이터베이스를 생성하기 위해서도 유니코드 수용은 필수적이라고 할 수 있다.

5.2. 서울대학교 도서관 유니코드 체계 구축을 위한 현 시스템 검토

5.2.1. 서울대학교 도서관 현행 시스템의 유니코드 지원 유무 현황

구분	세부 내용	수량	유니코드 지원(O/X)	비고
서버	UNIX(Solaris 2.8)	6대	O	
	NT(Win 2000 Ser)	5대	O	
	Linux	2대	O	인터넷 디스크용 및 게시판 서버
클라이언트 PC (업무용*)	Win XP	20대 (추정)	O	
	Win 2000		△	완벽하게 지원 못함
	Win 95/98/ME 등		X	
DBMS	Oracle 9i		O	
검색엔진	BRS Search 6.3		X	
프로그램 관련	Visual Basic 6.0		O	
웹	Tomcat 3.23		O	
	Apache 1.3.12		O	

표 28. 유니코드 지원 유무 현황

5.2.2. 유니코드 체계 구축을 위한 시스템 구비

기존 SOLARS II 시스템에서 사용하던 BRS Search는 유니코드를 지원하지 않기 때문에 다른 검색엔진으로 교체하는 방안을 마련해야 하며, 업무용 PC들도 유니코드를 지원하는 운영체제로 업그레이드해야 한다. 또한, 소프트웨어적인 측면에서 유니코드를 지원하지 않는 VB 6.0으로 개발된 클라이언트 등은 유니코드를 수용할 수 있는 Java,NET 등으로 전환해야 한다.

5.3. KS C 5601:1987자료의 유니코드 전환 시 고려 사항

5.3.1. 동형이음 한자 데이터 변환

유니코드에서 한자들을 CJK Ideographs Area에 배당하면서도, KS C 5601:1987과의 호환을 위해서 CJK Compatibility Ideographs 영역을 마련하고, 그곳에 동형이음 한자들을 배치해 놓았다. 따라서, 이와 같은 자료들을 변환하기 위해서는 두 가지 방법들을 고려해 볼 수 있다.

첫째, 동형이음한자의 다른 음가를 가진 코드를 서로 다른 코드 포인트에 배당하여 모두 코드체계에 수용하는 방법으로 아래의 표와 같다. 이 방식은 한자를 정확하게 입력하며, 한자의 한글음을 기계적으로 부여 가능하고, 정렬 가능하다는 장점이 있다. 하지만, 다른 음으로 검색했을 경우(예를 들어 요산요수(樂山樂水)를 낙산낙수(樂山樂水))는 검색이 되지 않고, 중국, 일본 한자와 호환 안 된다는 단점이 있다.

글자	KS C 5601:1987	Unicode	비고
악할 악(惡)	E4C2	60E1	CJK Ideographs(3400~9FFF)
미워할 오(惡)	E7F7	F9B9	CJK Compatibility(F900~FAFF)

표 29. 동형이음한자를 서로 다른 유니코드 영역에 배당하는 경우

둘째, 서로 다른 음 중 대표적인 음 하나만을 코드체계에 수용하는 방식으로, 유니코드 CJK Ideographs Area에 있는 대표음만을 수용하는 것이다. 이 방식은 음가에 상관없이 하나의 한자로 변환하여 추가 비용을 줄일 수 있고, 중국/일본 한자와도 호환가능하다는 장점이 있지만, 발음에 의한 정렬이 힘들고 동일한 문자를 각 나라마다 다르게 인코딩할 수 없다는 단점이 있다.

글자	KS C 5601:1987	Unicode	비고
악할 악(惡)	E4C2	60E1	CJK Ideographs(3400~9FFF)
미워할 오(惡)	E7F7	60E1	CJK Ideographs(3400~9FFF)

표 30. 동형이음한자를 같은 유니코드 영역에 배당하는 경우

이 두 가지 의견 중에서, 국립중앙도서관은 이론적으로는 동형이음한자를 CJK Ideographs Area로 변환하는 것을 추천하면서도, 유니코드 데이터를 KS C 5601:1987로 변환하여 많이 사용하는 향후 몇 년 동안은 CJK Compatibility Ideographs Area와 함께 변환하는 첫 번째 방법을 추천하고 있다.

따라서, 본 연구는, 국립중앙도서관의 분석과 마찬가지로, 기본적으로 KS C 5601:1987과 호환을 위해 마련된 코드를 이용하여 두 가지로 변환하는 것이 바람직하다고 생각된다. 또, 첫 번째 안을 채택했을 경우, 발생하는 검색 문제는 CJK Compatibility Ideographs Area에 배당되어 있는 코드 포인트들의 수가 361여개 밖에 되지 않으므로 별도의 매핑 테이블을 고안하는 것이 낫다고 생각된다. 또한, 검색 시에도 단순한 매핑 테이블만으로도 검색 및 변환이 가능하다.

[부록 3]은 C 언어로 구현한, CJK Compatibility Ideographs Area와 CJK Ideographs Area의 매핑 테이블을 출력하는 프로그램이다.

5.3.2. 외부기관과 데이터 교류 방안

도서관 시스템을 유니코드로 전환할 경우, 외부 기관과 데이터를 교환 시 유니코드 데이터들을 주고 받을 때는 문제가 생기지 않으나 현재 대부분의 시스템이 KS C 5601:1987로 인코딩된 데이터들을 사용하고 있으므로 자료의 손실이 발생할 수 있다. 이미 서울대학교 중앙도서관에서 분석한 것처럼, 데이터의 반

입 시에는 유니코드가 KS C 5601:1987 인코딩 시스템을 포괄할 수 있으므로 단순 변환 시스템을 구성할 수 있지만, 유니코드 데이터를 외부에 반출할 때 KS C 5601:1987에 없는 문자를 대체 문자로 변환하여 반출해야 한다.

이에 대한 대처방식으로 현 도서관 시스템에서 사용하고 있는 독자적인 대체 문자 표현 규칙이 있을 수도 있겠으나, 가급적 W3C의 HTML 문자 엔터티 표현 규약을 따르는 것이 좋다. HTML에서는 문자 엔터티를 &#와 ;사이에 십진수(decimal)로 기입하도록 하고 있다. 예를 들어, HTML 또는 XML에서 한글 '가'를 표현하기 위해서는 단순히 '가'라고 입력할 수도 있지만, 다음과 같은 방식으로 표현할 수도 있다. 다른 문자들도 마찬가지이다.

문자	16진수(hex)	10진수(dec)	엔터티 표현 (entity representation)
가	AC00	44032	가
惡	60E1	24801	惡
A	0041	65	A

표 31. W3C의 권고안을 따른 엔터티 표현의 예

이와 같은 W3C의 HTML 엔터티 표준 규약을 따를 경우, 자료들의 규칙적인 변환도 쉽고, 브라우저에 보여주기도 편하며, 이런 변환을 위해서 개발된 공용 컴포넌트들을 그대로 이용할 수 있다는 장점이 있다. 하지만, 본 문서에서 제시한 엔터티 표현 방식은 원론적인 논의로서, 기존 기관에서 사용하는 정보 공유 방식이 있다면, 정보 전달 및 검색엔진과의 호환성을 위해서 기존의 공유방식을 유지하는 것이 낫다.

5.3.3. 다국어 입력 방식

다국어를 입력하기 위해서는 Microsoft에서 개발한 IME 시스템을 그대로 이

용하는 방법과, 자체적으로 입력 응용프로그램을 개발하는 방법이 있다. 첫째, IME 시스템을 그대로 이용하면 개발 비용이 들지 않지만, 윈도우 2000 이상 지원해야 하고, 폰트가 없으면 입력 불가능하며 환경 설정을 사용자에게 요구한다는 단점이 있다. 둘째, 입력 프로그램을 자체 제작하면, 운영체제에 독립적으로 이용할 수 있지만 주기적인 개발비용이 증가한다는 단점이 있다.

두 가지 방법 중, 다국어 입력기를 독자적으로 개발하는 것은 여러 모로 비용이 많이 드는 일이다. Microsoft IME를 그대로 사용한다는 것은 사용자에게 불편을 초래할 수 있다. 하지만, 일반적으로, 사용자는 자신이 찾고자 하는 특정 언어를 많이 사용하는 경향이 있으므로, Microsoft IME의 환경 설정이 그리 문제가 되지 않는다. 따라서, 현재 운영체제에서 제공하는 Microsoft IME를 사용하는 것이 바람직하다.

5.3.4. 유니코드 Encoding 방식

유니코드의 인코딩 방식에 관해 국립중앙도서관에서 다음과 같은 의견을 제시하였다.

첫째, UNIX 운영체제 및 UNIX 어플리케이션 프로그램은 UTF-8을 기반으로 하는 것이 안정성이 뛰어나며, 둘째, Microsoft Windows 기반 시스템인 경우 UTF-16을 기본 인코딩 방식으로 사용하는 것이 좋으며, 셋째, 데이터베이스 관리 시스템은 각각의 컬럼에 대해 별도의 인코딩 방식을 설정할 수 있으므로 해당 데이터의 종류에 따라 인코딩 방식을 결정하는 것이 좋다.

위와 같은 원칙에 동의하면서도, 운영체제의 저장 공간이 얼마나 남아 있느냐를 반드시 고려해야 하고 하드웨어의 업그레이드 시기와 잘 맞추어야 한다.

5.3.5. 유니코드에서 지원하지 않는 한자 처리

현 시스템에서 유니코드에 포함되지 않는 한자들을 처리할 수 있는 방법은 없다. 이를 보완하는 방법으로, 한국역사정보통합시스템 및 국제 한자 표준화 기구에서 지정하는 코드 번호 등을 통일되게 부여하여 통일된 관리를 할 수 있는 방법을 마련해야 한다.

5.3.6. 유니코드의 한자가 한글음이 없는 경우 입력 방법

유니코드 입력기와 연관된 문제로, 또 다른 입력 방식(부수 및 획수) 을 개발해야 하거나, 유니코드 Ideograph들의 공식적인 로마나이즈 이름으로 입력할 수 있다.

5.3.7. 옛한글 문제

주리정(2001)은 현 유니코드 시스템에 옛한글의 자모만이 들어있고, 이에 대한 선정 기준도 모호하다는 단점을 지적한 바 있다. 옛한글의 음절을 구성하기 위한 자모만이 들어있고, 이에 대해 기본적인 음절구성 원리만이 들어있을 뿐, 이를 출력하기 위한 방법도 아직 마련되어 있지 않다. 따라서, 현재 옛한글을 효과적으로 표현하기 위해서는 유니코드 시스템에 옛한글 영역을 할당받기 위해 체계적인 조사 및 연구가 필요하다.

5.4. 서울대학교 도서관 유니코드 체계 적용 시 소요 일정 및 예산

5.4.1. 유니코드 개발 예상 소요 일정

구 분	M+1	M+2	M+3	M+4	M+5	M+6	M+7	M+8	M+9	M+10
시스템 분석 및 설계										
운영체제 변환										
어플리케이션 업그레이드										
검색엔진 도입										
업무용PC 업그레이드										
DB 변환 및 교열										
시범운영 및 가동										

표 32. 유니코드 개발 예상 소요 일정

5.4.2. 소요예산

(단위 : 백만원)

항 목	예상 비용	비고
Windows XP 업그레이드	2	20대x0.1만원 (OS 업그레이드)
유니코드 검색엔진 도입	100	
어플리케이션 업그레이드	521	
스토리지 구매	100	유니코드 변환 시 용량 확장
DB변환 및 교열	133	

표 33. 유니코드 개발 소요 예산

5.4.2.1. PC OS 업그레이드

유니코드 기반의 시스템이 원활히 운영되기 위해서는 업무용 PC는 Window XP로 반드시 업그레이드되어야 한다. 다행히도 대부분의 업무용 PC는 XP를 사용하고 있으며, 분관의 일부 PC와 학과 및 연구소 PC 중에서 XP로 OS 업그레이드되어야 할 PC는 대략 20여대 일 것으로 추정된다.

5.4.2.2. 유니코드 검색엔진 도입

도서관에서 현재 검색엔진으로 사용하고 있는 BRS는 유니코드를 지원하지 않으므로, 유니코드기반의 시스템에서 구동할 수 있는 검색엔진을 새로 도입해야 한다. 유니코드를 지원하는 검색엔진으로는 Verity K2, 다셀 21, Konan Docruzer, Search Formula-1(Korea WISEnut) 등이 있다. 이들 검색엔진 도입 시에는 성능테스트(BMT)를 통해 도서관의 데이터의 특성과 이용자 이용행태를 반영할 수 있는 검색엔진을 도입해야 한다.

5.4.2.3. 어플리케이션 업그레이드

어플리케이션 업그레이드 대상은 SOLARSII의 수서시스템, 연속간행물시스템, 기증·교환시스템, 목록시스템, 검색시스템, 대출/반납시스템, 장서관리시스템, 상호대차시스템, 경영정보시스템, 도서비관리시스템, 권한관리시스템, 통합검색시스템, 이용자관리시스템, 기사색인시스템과 전자도서관시스템을 대상으로 한다. 어플리케이션 업그레이드 비용은 Man/Month로 다음과 같이 예측하였다.

(단위 : 원)

구 분	산 출 내 역	금 액	비 고
직접인건비	고급기술자 176,998원x0.5명x25일x10개월 중급기술자 142,921원x1.5명x25일x10개월 초급기술자 112,530원x4명x25일x10개월	188,250,125	22,124,750 53,595,375 112,530,000
직접경비	-	-	
제경비	직접인건비 x 110%	207,075,138	
기술료	(직접인건비+ 제경비) x 20%	79,065,053	
부 가 세		47,439,032	
[redacted]			

표 34. 어플리케이션 업그레이드 비용

5.4.2.4. 스토리지 (Storage)

도서관의 전체 데이터가 유니코드로 변환되면서 데이터 사이즈가 2배 이상으로 커질 것으로 예상되는 바, 스토리지가 추가 구매되어야 한다.

구분			용량	비 고	
EMC Symmetrix 8530	현재	사용	150GB	사용용량 (150GB)	오라클 DB (50GB)
		가용	150GB		색인 (70GB)
		전체	300GB		기타 (30GB)
	유니코드 변환 시	전체	400GB	· 현재 DB 사용용량의 2배 이상 필요 (300GB + 추후 DB 증가 고려) · 스토리지에 400GB 디스크 확장 필요	

표 35. 스토리지 추가 구매 예상 비용

5.4.2.5. DB 변환 및 교열

SOLARSII 시스템 내의 MARC 서지데이터를 비롯하여, 이용자DB, 수서DB, 예산DB 등 방대한 양의 데이터가 유니코드로 변환되어야 하고 더불어 전자도서관 내의 XML 메타데이터와 관련 운영 데이터들이 모두 유니코드로 변환되어야 한다. 전체 데이터의 일괄 유니코드로의 변환과 아울러 대체문자는 정확한 유니코드 값으로 대체되어야 한다. 기본적으로 프로그램으로 일괄변환을 하고, 이후 교열작업을 통해서 데이터 변환에 따른 오류를 줄여나가야 한다.

이에 대한 구체적인 작업 내용 및 소요 예산은 다음과 같다.

구분	대상 물량(건)	작업내용 (일처리건수)	예상 비용(천원)
1) 전체 DB 일괄 변환 ● 전체 DB변환 ● 대체입력문자 변환	11,493,473		100,000
2) 대체입력문자 교열	250,642	교열 (400건)	24,409
3) 수작업 변환	22,782	입력 및 교열 (100건)	8,874
[illegible]	[illegible]	[illegible]	[illegible]

표 36. 데이터베이스 변환 및 교열 예상 소요 비용

첫째, 전체 DB 일괄 변환은 SOLARSII 시스템 내 MARC서지데이터와 관련 운영데이터, 전자도서관의 XML 메타데이터와 관련 운영데이터 등 전체적으로 데이터 변환을 위한 프로그램 개발 적용, 교열, 검수작업까지 포함한 내용이다. 그러나 정확한 대상물량 산출이 어렵기 때문에, 산출이 가능한 서지DB, 전거DB, 기사색인DB, 컨텐츠메타DB만을 대상으로 하여 서지DB 1,227,676건/ 전거DB 244,496건/ 기사색인DB 9,615,098/ 원문DB 406,203건(2003년 정보화 현황책자 참고)으로 추정하였다.

둘째, 대상물량 250,642건은 MARC서지 데이터 중에 프랑스어(28,556건) 독일어(76,770건) 스페인어(5,315건) 중국어(3,770건: 전체량 중 10% 예상) 일본어(136,231건)를 추산해서 산출하였으며, 또한 자료의 특성상 해당 언어별 전문가를 채용하여 교열작업을 수행하도록 계획하고 있다. (교정자 38,955원*25일 기준)

셋째, KS C 5601:1987에서 지원하는 한자가 없어서 한글이나 다른 문자로 대체 작성하여 MARC의 890 태그에 표기된 자료가 22,782건이고, 이 자료는 실제 자료를 확인해서 데이터에 수작업으로 입력해야 한다.

6. 결 론

유니코드는 KS C 5601:1987을 비롯한 기존의 문자집합이 가지고 있었던 다국어 환경에서의 문제점들을 극복하고, 이들을 대체하기 위해 개발되었다. 유니코드는 컴퓨터에 저장하려는 모든 문서 텍스트들을 인코딩하기 위한 방법을 제공하기 위한 국제 표준으로, 오늘날 실제 사용되는 모든 문자들과, 학술기호, 수학기호, 언어학기호, 프로그래밍 기호들을 망라하고 있다. 비록 기술적인 문제와 비판도 있지만, 유니코드는 컴퓨터의 주요 인코딩 체계로 부상했다.

다국어 환경을 지원하기 위한 국외 기관들의 유니코드 체계 검토에 발 맞추어, 국립중앙도서관, 한국교육학술정보원 등, 여러 국내 기관들도 유니코드 체계 전환을 검토해 왔다. 특히, 서울대학교 도서관은 전문 학술자료 및 외국 자료를 다량 보유하고 있으며, 이를 위한 고품질 서지 데이터베이스를 작성 및 관리해야 한다. 이와 같은 다국어 자료들을 관리하려면 KS C 5601:1987 코드 체계로는 한계가 있어서, 서울대학교 도서관에서는 임의의 기호 규칙을 만들어 다국어 데이터를 구축해 왔으며, 이를 극복하기 위해서는 유니코드 도입이 절실하다. 서울대학교 도서관은 향후 대학 도서관들이 유니코드 체계로 전환하는데 따른 표준 모델을 제시하고, 유니코드 체계로 전환한 대학 도서관의 목록을 수용할 수 있도록 미리 대처해야 한다. 또한, 국외 학술 정보 서비스 기관들과의 목록 상호 교환을 통한 정보 서비스의 확대도 준비해야 한다.

도서관 시스템이 완전한 유니코드 체계를 지원하기 위해서는 소장 자료의 정보가 유니코드로 저장이 가능하고, 유니코드로 입력, 편집, 검색이 가능하여야 한다. 또한, 목록, 상호 대차, 원문 서비스 등 각종 도서관 서비스가 유기적으로 연동되어야 한다. 그리고, 외부 도서관과의 정보 교환을 위해서 유니코드 MARC 반입, 반출을 지원해야 한다.

본 연구에서는 서울대학교 도서관을 유니코드 체계로 전환하는데 필요한 기술적 환경, 서지 목록 정보, 응용 프로그램 환경 등을 종합적으로 검토하였으며, 여러 면에서 유니코드로의 전환에 필요한 제반 여건이 갖추어졌음을 알 수 있었다.

첫째, 기술적인 면에서 많은 운영체제와 데이터베이스 시스템들이 유니코드를 지원하고 있다. 사용자 PC에서도 현재 사용자가 가장 많은 Windows XP는 유니코드를 완벽히 지원하고 있고, Oracle9i와 Microsoft SQL 2000 등의 데이터베이스 시스템도 자료의 유니코드화가 가능하다.

둘째, 서울대학교 도서관 서지 목록들은 다국어를 많이 포함하고 있어서, KS C 5601:1987에서 표현하지 못하는 문자들은 임의의 기호들로 변환하였다. 따라서, 이를 유니코드로 변환하기 위해서는 일괄 프로그램을 작성하여 변환하되, 동형이음한자, 유니코드에서 지원하지 않는 한자, 인코딩 방식, 다국어 입력 방식 등은 본 연구에서 제시한 원칙을 사용하여야 할 것이다.

셋째, 도서관 시스템을 유니코드로 전환할 경우, 외부 기관과 데이터를 교환시 유니코드 데이터들을 주고 받을 때는 문제가 생기지 않으나, 현재 대부분의 시스템이 KS C 5601:1987로 인코딩된 데이터들을 사용하고 있으므로 자료의 손실이 발생할 수 있다. 유니코드 데이터를 외부에 반출할 때는 KS C 5601:1987로 표현할 수 없는 문자를 해당 기관과 협의를 거쳐 반출해야 한다.

서울대학교 도서관의 유니코드 체계로의 전환이 국내 도서관시스템의 유니코드 전환의 모델을 제시하기를 바라며, 본 연구를 바탕으로 효율적인 유니코드 전환이 이루어지길 기대한다.

참고 문헌

김경석(1999), 컴퓨터 속의 한글 이야기 둘째보따리, 부산대학교 출판부

김경석(1995), 컴퓨터 속의 한글 이야기, 영진출판사

김태수 외(1996), 기계가독목록의 이해, 문헌정보처리연구회

류종범 외(2002), 과학기술정보유통시스템 개발, 한국과학기술정보연구원

오동근 역(2001), MARC의 이해, 태일사

이경호, 김정현(2003), 자료목록별 KORMARC/MARC21을 중심으로, 대구대학교 출판부

국립중앙도서관(2003), 국립중앙도서관 표준화 연구중 단일문자 표준 연구, 한국전산원

주리정(2001), 유니코드 구조와 문제점, 한국정보관리학회 제8회 학술대회논문집

최석두(1997), 대학도서관 분담편목용 입력기본 표준에 관한 연구, KRIC

한국어 정보처리연구소(2001), "C로 구현한 한글 코드 시스템 프로그래밍 가이드", 도서출판 골드

한혜영(1998), 국내서 단행본 입력 방안에 관한 연구, KRIC

About search system, K2.
<http://www.3soft.com>

Building International Application with Power Builder
<http://www.sybase.com/detail>

Guide to Developer's Kit for Unicode, 1997.
<http://www.sybase.com/onlinebooks>

Forms of Unicode, IBM developer and President of the Unicode Consortium, IBM 1 September 1999.
<http://www-106.ibm.com/developerworks/library/utfencodingforms/>

General Information About the Arial Unicode MS font.
<http://support.microsoft.com/default.aspx?scid=kb;en-us;287247>
Internationalizing an Application.
<http://manuals.sybase.com/onlinebooks/group-pb/pbg0900e/apptech>
Library of Congress(2000), MARC21 Specifications for Record Structure, Character Sets, and Exchange Media, Library of Congress Network Development and MARC Standard Office, Web Version.
<http://www.loc.gov/marc/specifications/spechome.html>
MARBI(1996), MARC Proposals No. 96-10, USMARC Character Set Issues and Mapping to Unicode/UCS,
<http://www.loc.gov/marc/marbi/1996/96-10.html>
MARBI(1997), MARC Proposals No. 97-10, Use of the universal code character set,
<http://www.loc.gov/marc/marbi/1997/97-10.html>
MARBI(1998), MARC Proposals No. 98-18, Unicode Identification and Encoding,
<http://www.loc.gov/marc/marbi/1998/98-18.html>
Microsoft(2000), MSDN help for Microsoft Visual Studio 7.0,
<http://www.microsoft.com/msdn>
Microsoft, Develop Unicode Applications for Windows 9x Platforms with the Microsoft Layer for Unicode,
<http://msdn.microsoft.com/msdnmag/issues/01/10/MSLU/default.aspx>
Microsoft, Overview of Windows XP International Support
<http://www.microsoft.com/globaldev/handson/dev/winxpintl.mspx>
Microsoft, The Microsoft Layer for Unicode on Windows 95/98/Me

Systems,
<http://www.microsoft.com/globaldev/handson/dev/mslu_announce.mspx>
Microsoft, Windows XP/2000 default system font,
<http://www.microsoft.com/globaldev/DrIntl/columns/017/default.mspx#Q2>
NII(2000), 圖書館 システムの多言語對應, 平成12年度 신CAT/ILLシステム 說明資料-7.
NII(2001), NACSIS-CAT, 目錄システムの多言語對應, 第2版
NII(2001), はじめての新CAT/ILL, 第2版
NII(2001), NACSIS-CAT/ILL ニュースレター5号.
NII(2001), NACSIS-CAT/ILL ニュースレター8号.
NII(2001), NACSIS-CAT/ILL ニュースレター9号.
OCLC(2001), OCLC Annual Report, 2000/2001.
OCLC(2002), OCLC NewsLetter, July.
OCLC(2002), OCLC Annual Report, 2001/2002.
Oracle(2000), Migration Oracle Internet Applications to Support Unicode, An Oracle Technical White Paper.
Oracle(2000), Oracle8i Release 3 National Language Support.
Oracle(2001), Best Practices for Globalization using the Oracle 9i Internet Application Server, Version 1.0.
Oracle(2002), Globalization Support Oracle Unicode database support, An Oracle White Paper.
Oracle(2002), Migration to Unicode Datatypes for Multilingual Databases and Applications in Oracle9i, An Oracle White Paper.
Oracle(2002), Oracle9i Database Globalization Support Guide, Release 2(9.2), Part No. A96529-01.
PowerBuilder 8.0.2 Maintenance Release New features.

<http://www.justinfo.com/pb802.asp>

Richard Gillam(2003), Unicode Demystified, A Practical Programmer's Guide to the Encoding Standard, Addison Wesley.

The Unicode Consortium(2003), The Unicode Standard Version 4.0.

Unicode and Multilingual File Conversion, Font and Keyboard Utilities for Windows Computers.

<http://www.alanwood.net/unicode/utilities_fonts.html#globalime>

Unicode fonts for Windows computers

<http://www.alanwood.net/unicode/fonts.html>

Unicode Support in the Solaris Operating Environment

<http://www.sun.com/software/whitepapers/wp-unicode>

YAZ Toolkit

<http://www.indexdata.dk/yaz/>

다국어지원데이터베이스 : 1. 개념 및 아키텍처

<http://www.en-core.com/bin/main/module/solution/view.asp?solution_code=&searchString=&column=&article_id=13329&state=view&board_id=solution&page_num=3&group_id=19570&direction=n&step=0>

다국어지원데이터베이스 : 2. 데이터베이스 구축하기

<http://www.en-core.com/bin/main/module/solution/view.asp?solution_code=&searchString=&column=&article_id=13330&state=view&board_id=solution&page_num=3&group_id=19570&direction=n&step=0>

다국어지원데이터베이스 : 3. Linguistic 정렬과 인덱스

<http://www.en-core.com/bin/main/module/solution/view.asp?solution_code=&searchString=&column=&article_id=13331&state=view&board_id=solution&page_num=3&group_id=19570&direction=n&step=0>

다국어지원데이터베이스 : 4. Character Set Migration

<http://www.en-core.com/bin/main/module/solution/view.asp?solution_code=&searchString=&column=&article_id=13332&state=view&board_id=solution&page_num=3&group_id=19570&direction=n&step=0>

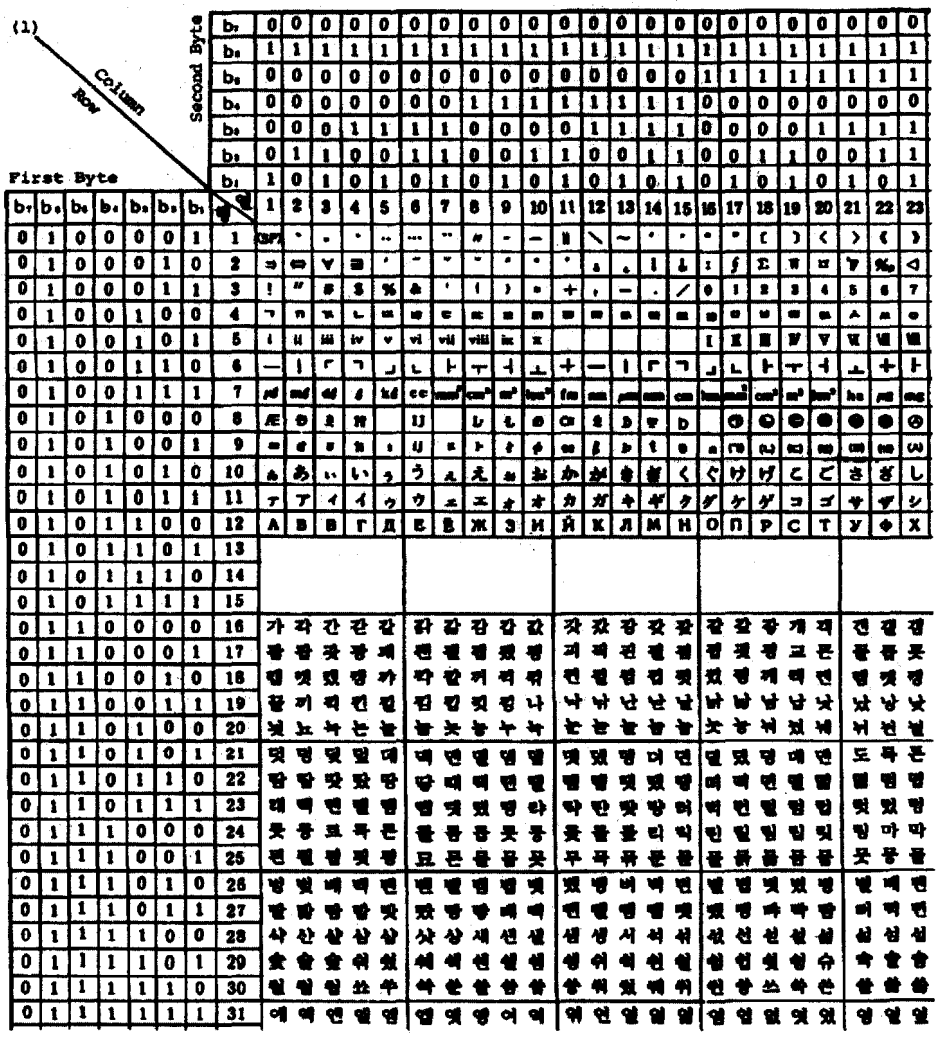
부록 1. ISO 2022에서 사용하는 문자 세트에 대한 용어의 정의들

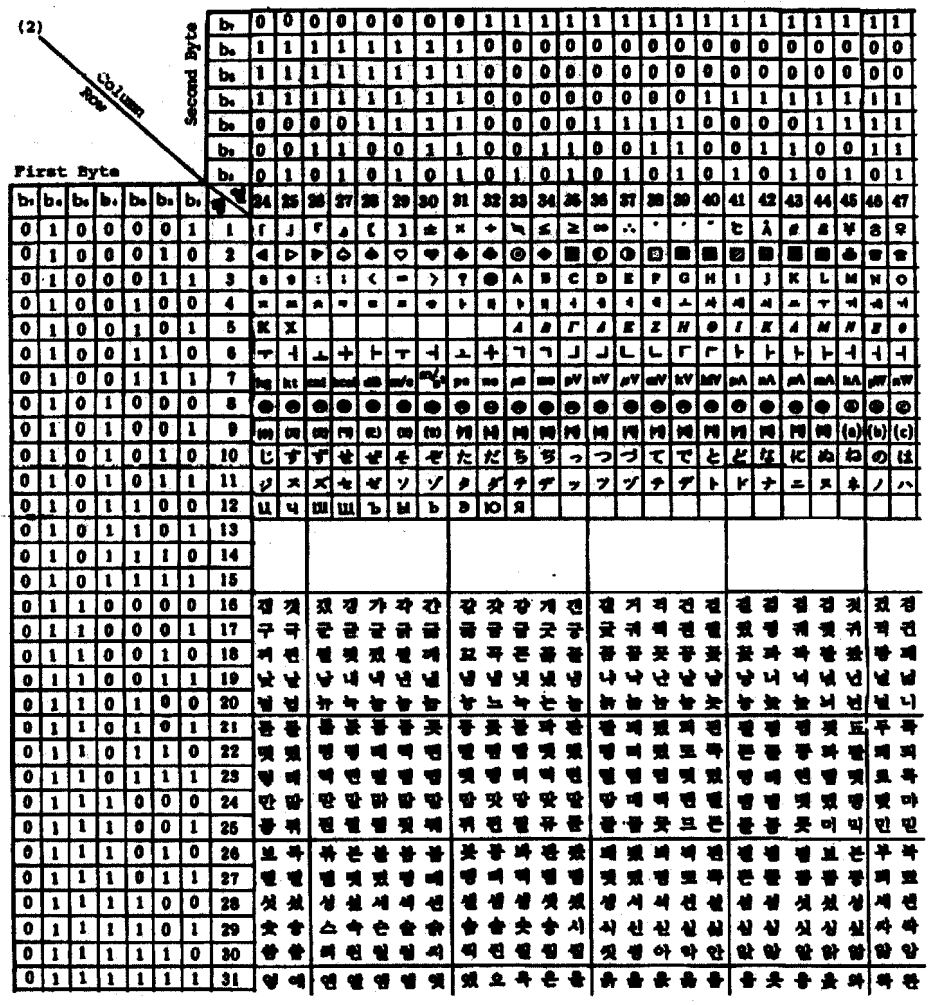
- **Bit Combination** : An ordered set of bits used for the representation of characters
- **Byte** : A bit string that is operated upon as a unit (includes 7-bit, 8-bit, and 16-bit "bytes," octet equals 8 bits)
- **Coded character set; code** : A set of unambiguous rules that establishes a character set and the one-to-one relationship between the characters of the set and their bit combinations.
- **Code table** : A table showing the character allocated to each bit combination in a code
- **Code extension** : The techniques for the encoding of characters that are not included in the character set of given code
- **Combining character** : A member of an identified subset of a coded character set, intended for combination with the preceding or following graphic character, or with a sequence of combining characters preceded or followed by a non-combining character
- **Control character** : A control function the coded representation of which consists of a single bit combination
- **Control function** : An action that affects the recording, processing, transmission or interpretation of data, and that has a coded representation consisting of one or more bit combinations
- **Designate** : To identify a set of characters that are to be represented, in some cases immediately and in others on the occurrence of a further control function, in a prescribed manner
- **Escape sequence** : A string of bit combinations that is used for control purposes in code extension procedures; the first of these bit

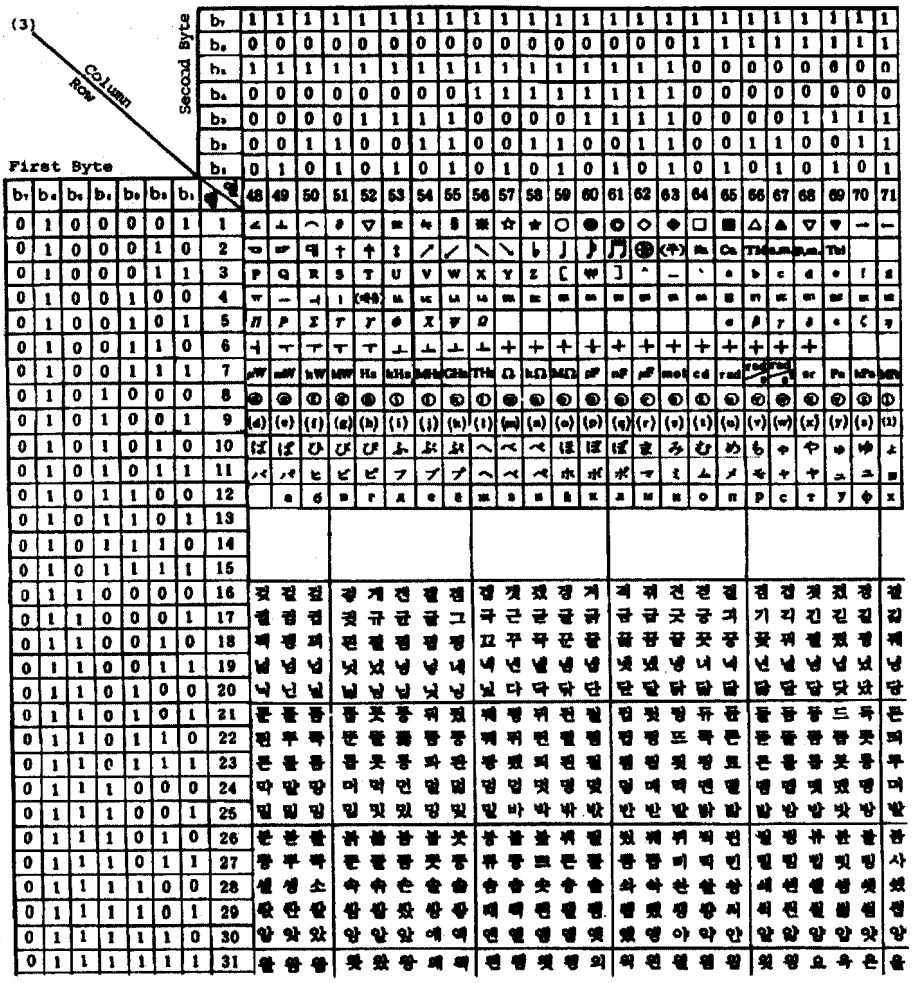
combinations represents the control function ESCAPE

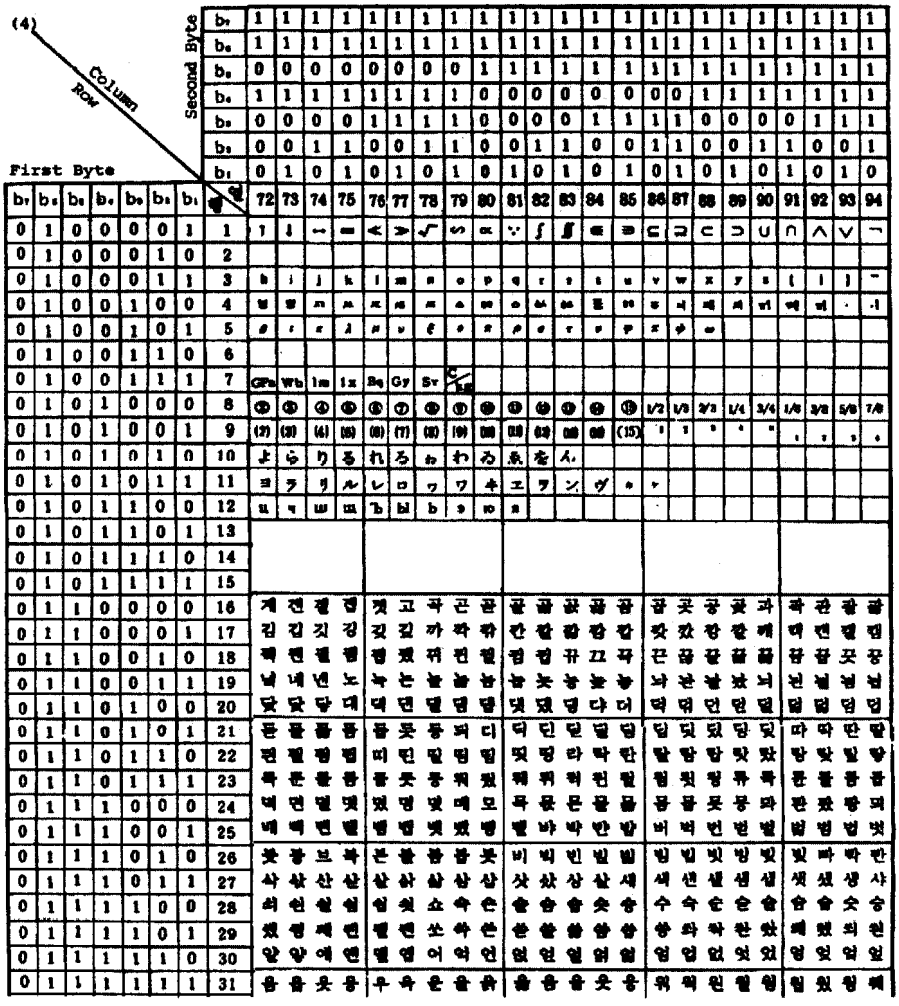
- **Graphic character** : A character, other than a control function, that has a visual representation normally handwritten, printed or displayed, and that has a code representation consisting of one or more bit combinations
- **Invoke** : To cause a designated set of characters to be represented by one or more bit combinations of a coded character set
- **Repertoire** : A specified set of characters that are each represented by one or more bit combinations of a coded character set

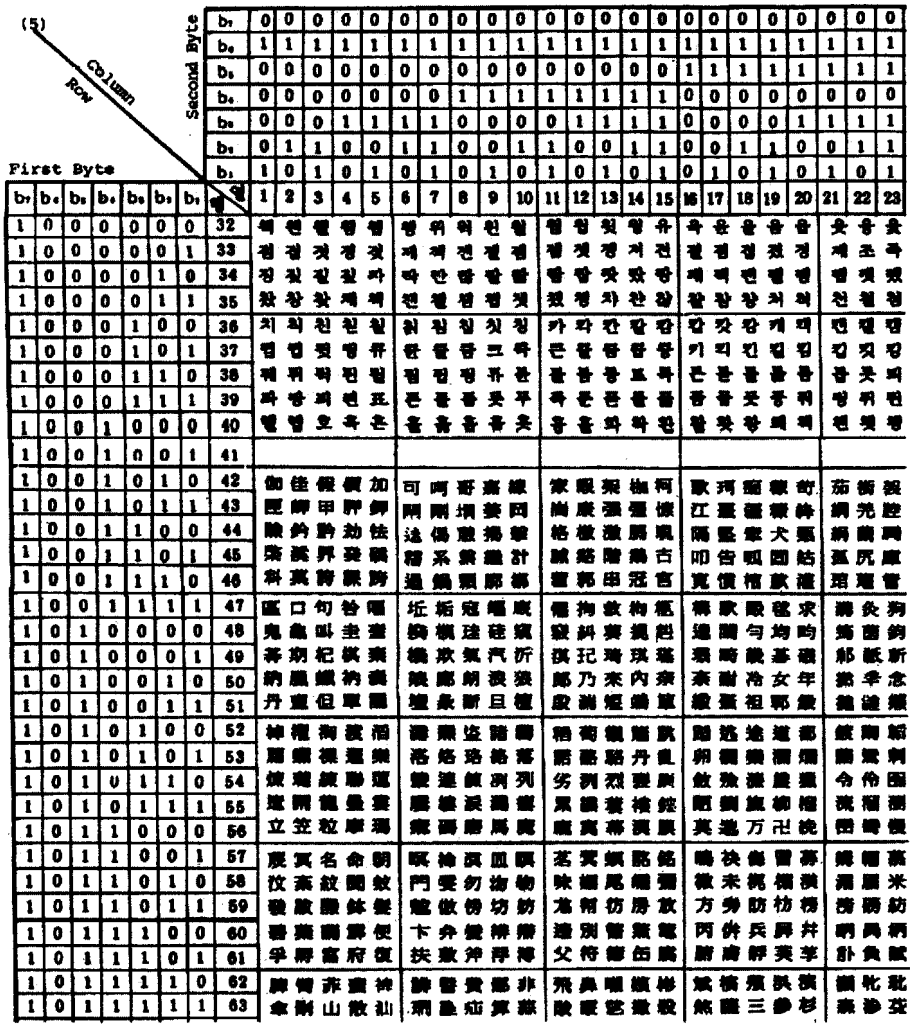
부록 2. KS C 5601:1987 문자표



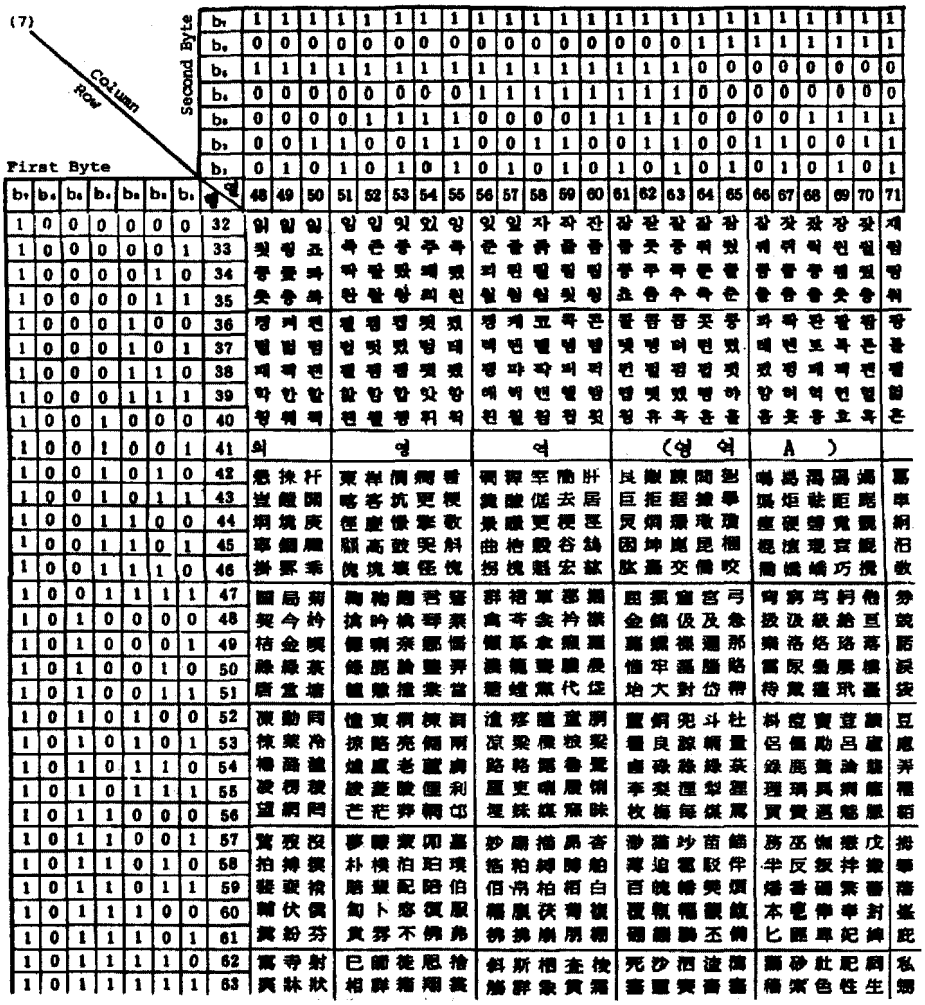

(2)
Column
Row
Second Byte
First Byte

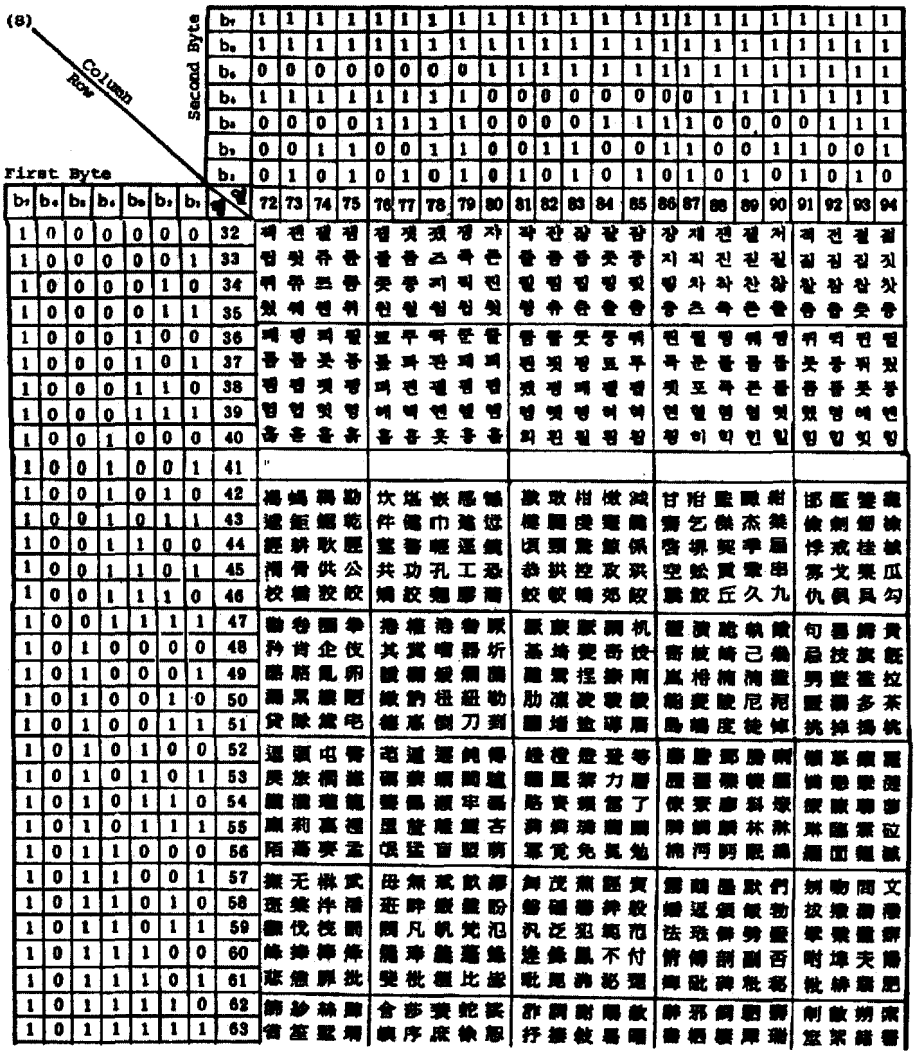

(3)
Column
Row
Second Byte
First Byte


(4)
Column
Row
Second Byte
First Byte


(5)
Column
Row
Second Byte
First Byte

(6) Row / Column								Second Byte																				
								b_7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
								b_6	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
								b_5	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
								b_4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1
								b_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1
								b_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
First Byte								b_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1			24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1	0	0	0	0	0	0	32		오	옥	은	을	을	을	을	옷	을	을	을	을	을	을	의	원	웹	웹	이	익	인	일
1	0	0	0	0	0	1	33		은	을	을	을	을	옷	에	에	웹	웹	을	을	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹
1	0	0	0	0	1	0	34		웹	와	웠	정	제	제	에	에	웹	웹	웹	웹	웹	웹	웹	웹	조	웹	은	을	을	을
1	0	0	0	0	1	1	35		정	웠	정	제	제	에	에	웹	웹	웹	웹	웹	제	원	웠	제	원	웹	웹	웹	웹	웹
1	0	0	0	1	0	0	36		웹	웠	정	와	와	와	와	와	와	와	원	원	웹	웹	웠	웹	웹	웹	웹	웹	웹	웹
1	0	0	0	1	0	1	37		와	와	와	와	와	와	와	와	와	와	와	와	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹
1	0	0	0	1	1	0	38		웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹
1	0	0	0	1	1	1	39		웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹	웹
1	0	0	1	0	0	0	40		의	적	원	웹	웹	웹	웹	웹	은	을	을	웹	우	우	은	을	을	웹	웹	웹	웹	웹
1	0	0	1	0	0	1	41				사					을					자				경					
1	0	0	1	0	1	0	42		買	豊	賠	弦	管	珠	槐	劍	腰	深	槐	刹	壁	壇	蜂	光	關	嶺	京	言	洗	干	幹
1	0	0	1	0	1	1	43		阿	紅	見	得	題	賠	賠	賠	何	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠
1	0	0	1	1	0	0	44		賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠
1	0	0	1	1	0	1	45		賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠
1	0	0	1	1	1	0	46		賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠	賠
1	0	0	1	1	1	1	47		球	絹	賃	枯	道	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	0	0	0	48		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	0	0	1	49		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	0	1	0	50		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	0	1	1	51		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	1	0	0	52		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	1	0	1	53		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	1	1	0	54		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	0	1	1	1	55		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	0	0	0	56		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	0	0	1	57		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	0	1	0	58		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	0	1	1	59		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	1	0	0	60		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	1	0	1	61		劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	1	1	0	62		貨	實	杉	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍
1	0	1	1	1	1	1	63		夢	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍	劍


(7)
Column
Row
Second Byte
First Byte



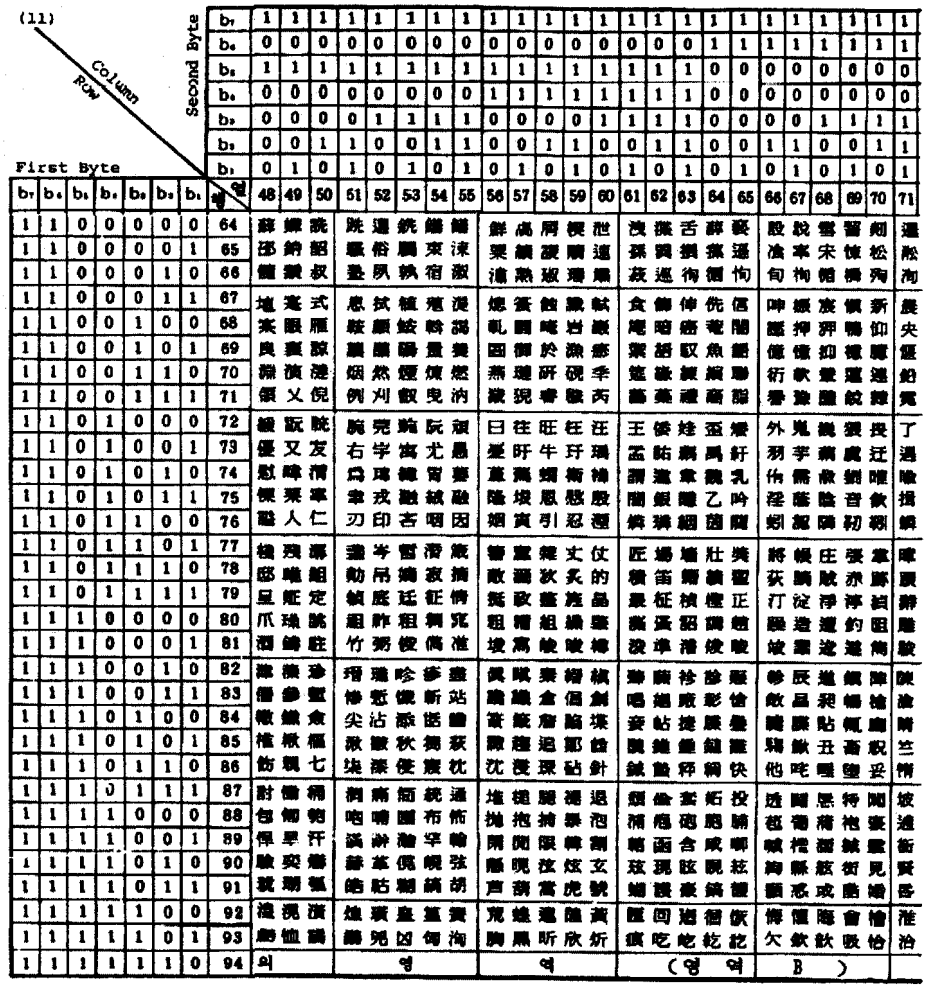
(9)

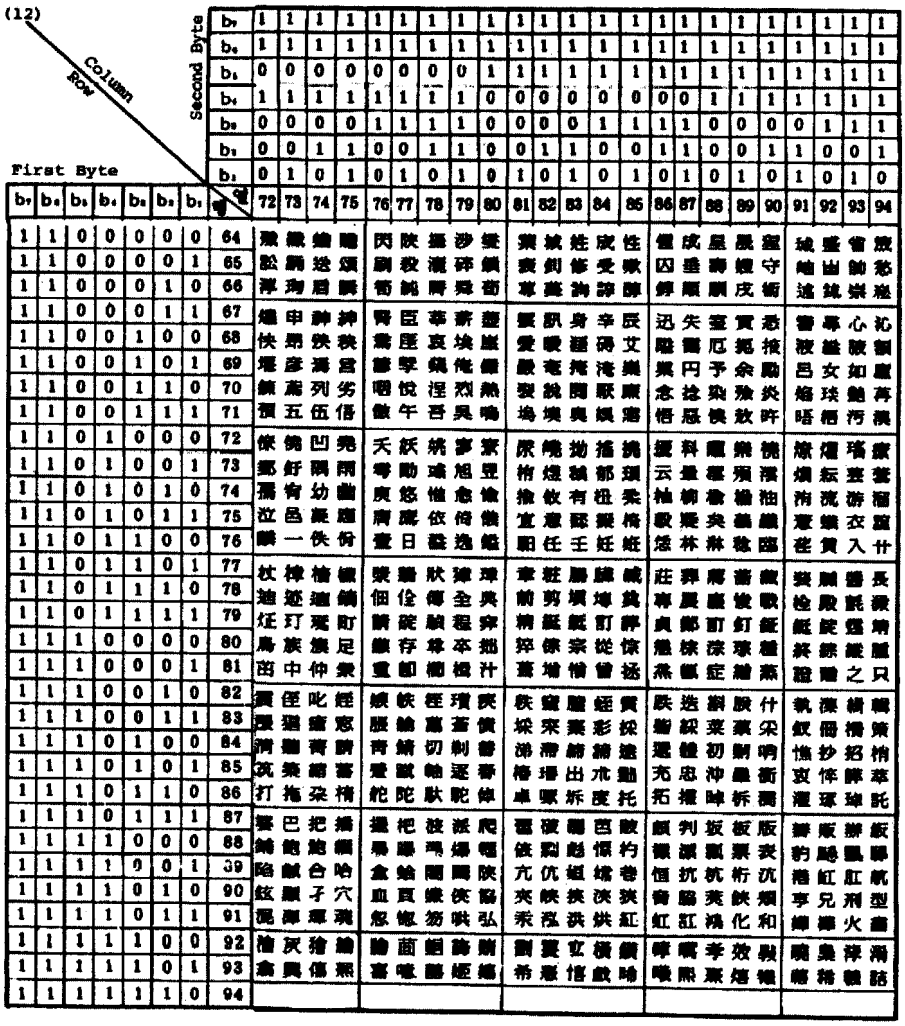
Column / Row

Second Byte

b_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b_6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
b_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
b_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
b_3	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1
b_2	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
b_1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
First Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1 1 0 0 0 0 0	64	荷	舒	春	西	雪	逝	働	乗	寒	爽	席	惜	昔	哲	折	汐	新	溝	石	碩	庫	輝
1 1 0 0 0 0 1	65	聖	舒	椿	誠	設	世	勢	剰	殊	笹	細	説	獣	召	噛	塑	親	小	少	巣	所	捕
1 1 0 0 0 1 0	66	成	手	腿	捜	収	数	労	樹	残	巣	縄	狩	猷	踊	瑳	痩	睡	秀	幕	要	秤	殺
1 1 0 0 0 1 1	67	嵐	惹	瞭	蒼	装	拾	酉	個	丞	乗	僧	勝	升	承	昇	縄	縄	陸	待	老	始	新
1 1 0 0 1 0 0	68	沈	深	暢	霜	荘	舎	什	十	蝿	氏	亞	俄	兒	野	峨	峡	我	牙	芽	若	鋳	集
1 1 0 0 1 0 1	69	様	器	軟	霧	葬	捨	治	夜	餘	師	爺	耶	若	鯨	放	漿	賂	約	域	薬	蒋	層
1 1 0 0 1 1 0	70	旅	軟	腿	妹	霊	赦	煽	与	令	興	事	間	嶺	影	跌	漿	瞑	力	植	柔	役	泳	漢
1 1 0 0 1 1 1	71	旗	腿	然	釈	霊	繁	畑	予	領	営	塁	嶺	裕	怜	映	裟	賂	亦	植	楽	永	漢	
1 1 0 1 0 0 0	72	烏	然	猷	矣	鳳	裏	焼	屋	沃	獄	玉	鉱	温	竜	瘟	磯	鵬	鷹	冗	専	覚		
1 1 0 1 0 0 1	73	窃	猷	猶	鉱	鶏	褒	蛾	要	鉱	遙	玉	遣	鍋	慾	欲	裕	蝶	鷹	得	悪	冥	専	
1 1 0 1 0 1 0	74	運	猶	醤	献	顕	褒	蛾	繁	鉱	原	遣	員	鍋	裕	浴	蝶	鷹	寛	獲	遊	冥	専	
1 1 0 1 0 1 1	75	潮	醤	二	以	顕	繁	畑	癌	鉱	越	員	奥	鍋	怡	欲	裕	蝶	鷹	得	悪	冥	専	
1 1 0 1 1 0 0	76	潮	醤	二	以	顕	繁	畑	癌	鉱	越	員	奥	鍋	怡	欲	裕	蝶	鷹	得	悪	冥	専	
1 1 0 1 1 0 1	77	立	笠	粒	仍	剛	孕	彷	仔	吾	始	委	子	字	孜	志	慈	滋	炙	衆	症	症		
1 1 0 1 1 1 0	78	陣	両	現	在	宰	才	材	我	障	神	災	倖	暖	財	慈	寮	満	争	挙	双	浮		
1 1 0 1 1 1 1	79	煎	頂	建	制	畑	嚇	笠	堤	像	籍	砂	栓	梯	第	銘	寮	鍋	電	賜	鉦	鋒		
1 1 1 0 0 0 0	80	勝	頂	鍵	鍋	則	哺	左	座	捨	提	梯	満	侏	株	腸	賓	呪	雨	腸	賓	除		
1 1 1 0 0 0 1	81	隊	頂	鍵	鍋	佐	指	隼	支	昔	枚	釈	止	池	持	演	知	砥	滋	紙	紙	胎		
1 1 1 0 0 1 0	82	甩	旭	址	志	持	且	化	借	叉	雌	差	次	此	辻	御	茶	腰	超	遮	捉	着		
1 1 1 0 0 1 1	83	綿	集	腰	意	意	伺	化	剛	尺	威	拓	沖	茗	斥	癖	超	雌	酢	車	捉	千		
1 1 1 0 1 0 0	84	貴	楚	取	焦	態	萌	礎	錐	秒	肯	神	鹿	辞	草	癖	超	雌	酢	醜	促	嘱		
1 1 1 0 1 0 1	85	嫩	楚	取	吹	炒	萌	礎	軍	秒	就	神	鹿	辞	草	癖	超	仄	酢	醜	捉	嘱		
1 1 1 0 1 1 0	86	黄	取	吹	嘱	参	萌	欣	軍	就	肯	鹿	辞	草	露	癖	超	仄	酢	醜	促	修		
1 1 1 0 1 1 1	87	銅	呑	嘱	坦	弾	停	散	嚢	炭	統	寄	脱	探	吐	統	貪	塔	搭	揚	右	帯		
1 1 1 1 0 0 0	88	阪	八	叭	捌	電	哨	惨	敗	袖	披	被	鞭	鍋	肥	彫	参	必	踏	幅	便	備		
1 1 1 1 0 0 1	89	品	陣	棟	項	装	風	埃	飛	彼	被	徳	鋳	過	邪	解	今	答	遅	離	正	勤		
1 1 1 1 0 1 0	90	行	陣	洞	絞	華	咳	焚	廃	疎	皮	徳	鋳	過	那	解	今	答	遅	離	峰	勤		
1 1 1 1 0 1 1	91	形	洞	花	華	駿	鍋	貨	厚	疎	被	徳	鋳	過	那	盤	今	答	遅	離	峰	勤		
1 1 1 1 1 0 0	92	編	禾	花	華	駿	牌	貨	厚	郎	披	電	磯	鍍	朽	換	魚	宣	幻	動	換	歌	既	
1 1 1 1 1 0 1	93	又	骨	膠	華	駿	鍋	貨	厚	吼	被	電	磯	鍍	朽	環	返	助	動	堵	煥	無	無	
1 1 1 1 1 1 0	94																							

(10) Column / Row	Second Byte																													
	b_7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	b_6	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b_5	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b_4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	b_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1
	b_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1
	b_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
First Byte		24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47					
b_7 b_6 b_5 b_4 b_3 b_2 b_1																														
1 1 0 0 0 0 0	64	伽	仙	僭	先	善	嬋	宣	嚥	斂	旋	渲	燁	琥	琺	職	堵	癬	禪	線	箱	戾	腺	膳	船					
1 1 0 0 0 0 1	65	佳	昭	悦	沼	繕	渕	覧	妬	焼	誰	警	喩	痴	笑	諜	訓	集	陳	蹴	随	雌	折	適	茴					
1 1 0 0 0 1 0	66	嘉	嘯	嚴	栴	茶	蝉	夢	嶼	袖	隻	警	輸	違	選	銕	錬	繍	隋	隴	隨	難	需	須	苗					
1 1 0 0 0 1 1	67	綽	尸	屎	毘	市	弒	侍	旌	是	時	柿	柴	猜	矢	示	翅	蒔	蓍	試	詩	誠	詣	家	針					
1 1 0 0 1 0 0	68	野	阿	雅	錻	鵐	鍔	蝕	嶼	揚	堤	恕	傷	梁	楊	疎	羊	鄭	錆	安	礁	岸	接	雀	梁					
1 1 0 0 1 0 1	69	賜	兇	伴	剛	塚	凉	逆	填	琅	蛍	鍋	塁	年	鋭	樺	幹	煽	拶	鉱	沈	鎖	常	孤	梢					
1 1 0 0 1 1 0	70	昌	疫	諤	御	靱	娯	聯	玲	嗅	琅	硬	塁	禎	殿	幹	聆	捐	挨	迎	鈴	鎖	英	孤	梢					
1 1 0 0 1 1 1	71	頑	環	贔	屓	娯	鍔	舜	玲	瑛	琅	塁	叡	禎	殿	幹	聆	捐	英	詠	迎	鈴	鎖	孤	梢					
1 1 0 1 0 0 0	72	妻	塵	猿	嵐	雨	雷	禍	瓦	寫	侵	臥	蛙	蝸	訛	帝	完	宛	喜	椀	頑	玩	琉	碗	碗					
1 1 0 1 0 0 1	73	勇	墳	塘	容	鷹	電	棺	満	湧	侵	唐	鞆	用	月	繍	越	位	囲	踊	館	偶	龍	佑	偶					
1 1 0 1 0 1 0	74	浸	漑	愛	容	鷹	塊	電	噂	湧	青	陛	倫	鴛	尹	雛	高	鍋	飴	伸	館	虎	龍	武	備					
1 1 0 1 0 1 1	75	雑	艦	願	六	而	垣	艦	内	寡	青	極	貽	痴	里	離	輸	飴	囲	玩	館	危	龍	国	律					
1 1 0 1 1 0 0	76	病	移	曜	而	耳	艦	立	寡	墓	極	裡	貳	遥	里	離	飴	囲	匿	測	館	貴	龍	豆	実					
1 1 0 1 1 0 1	77	磁	紫	者	自	茨	薗	蒋	語	資	蛙	作	勺	嚼	祈	詐	灼	炸	請	鉾	苅	酌	雫	鵠	廚					
1 1 0 1 1 1 0	78	低	儀	咀	姐	底	鍋	節	補	郷	沮	酒	点	薄	値	答	酌	貯	蓮	著	丁	走	亭	鷺	偵					
1 1 0 1 1 1 1	79	敏	折	浙	繍	沮	兆	抵	占	站	店	斯	点	冊	見	箸	貼	帖	擢	乗	積	倒	停	照	偵					
1 1 1 0 0 0 0	80	鯛	題	齊	俎	注	節	洲	嘲	弔	影	描	鵜	早	袖	書	舟	蝕	條	跡	走	踏	潮	照	偵					
1 1 1 0 0 0 1	81	童	朱	柱	株	注	詰	洲	澍	住	珠	鵜	鵠	村	舳	舟	蝕	条	荘	跡	走	踏	潮	遇	酎					
1 1 1 0 0 1 0	82	至	芝	正	蜘	蛛	識	貴	胜	遅	直	蛇	鵜	織	職	色	喰	鐘	振	指	晋	振	蓉	珍	茶					
1 1 1 0 0 1 1	83	宰	舗	霊	蛛	泉	銕	廷	唾	環	賑	蛇	鵠	縞	縁	食	賓	鐘	娠	嘴	縄	娠	札	茶	集					
1 1 1 0 1 0 0	84	天	川	擅	蜘	寸	浅	邪	舛	環	鶏	践	息	銅	縄	阡	職	邸	哲	催	晋	最	振	螺	推					
1 1 1 0 1 0 1	85	墓	島	蝶	鳴	恥	村	治	塚	鶏	鶏	鹿	惜	鍾	縄	職	鍾	鍾	撮	嘴	蝶	撮	抽	動					
1 1 1 0 1 1 0	86	唯	鳴	嫌	時	台	梱	治	嫁	待	鶏	鹿	惜	鍾	鍾	職	鍾	鍾	撮	嘴	蝶	撮	則	致					
1 1 1 0 1 1 1	87	猪	鯛	兒	台	太	意	離	兎	汰	泰	苔	胎	呑	跆	邸	娶	鐘	宅	挪	揮	揺	肪	土	佈					
1 1 1 1 0 0 0	88	片	篇	編	翻	下	運	向	貶	坪	平	柄	荷	評	吠	賀	運	職	廃	異	学	虚	厳	吐	閉					
1 1 1 1 0 0 1	89	部	乏	遥	何	行	何	向	廃	是	和	郷	張	媛	吠	香	職	場	許	意	新	郷	寒	肢	鶴					
! : 1 1 0 1 0	90	伸	幸	杏	呼	存	互	行	納	姑	弧	背	戸	媛	唄	香	職	場	朔	新	郷	献	歌	軒	渡					
1 1 1 1 0 1 1	91	離	鞄	乎	互	理	執	運	好	括	愛	弁	唆	園	鳥	響	輝	厨	浩	虚	潮	猟	雷	狐					
1 1 1 1 1 0 0	92	恒	換	換	理	理	理	理	媒	理	理	理	理	園	鳥	響	輝	厨	浩	悦	輝	悩	偶	晩	況					
1 1 1 1 1 0 1	93	爆	業	劇	量	理	理	理	爆	理	理	理	理	園	鳥	響	輝	厨	浩	虚	輝	脳	縄	休	挑					
1 1 1 1 1 1 0	94			사					용					자					정											


(11)
Column
Row
Second Byte
First Byte


(12)
Column
Row
Second Byte
First Byte

부록 3. CJK Ideograph Area와 CJK Compatibility Ideograph Area간의 변환 테이블 및 변환 출력 프로그램

```
/*****************************************************************
 *
 * AUTHOR : Insik, Cho
 * DATE : Jan 19, 2005
 * NOTE : CJK Ideograph Area와 CJK Compatibility Ideograph Area간의
 *        변환 테이블 및 변환 출력 프로그램
 * E-Mail : iaminsik@gmail.com
 *
 *****************************************************************/
#include <stdio.h>

#define WORD unsigned short
#define NF_COMPAT 361

/*****************************************************************
 *
 * CJK Compatibility Ideographs Area와 CJK Ideographs Area간의
 * 쌍으로 구성된 변환 테이블 (0xF900 ~ 0xFA6A)
 *
 * 1. Pronunciation variants from KS X 1001:1998 (0xF900 ~ 0xFA0B)
 * 2. Duplicates from Big 5 (0xFA0C ~ 0xFA0D)
 * 3. The IBM 32 compatibility additions (0xFA0E ~ 0xFA2D)
 * 4. JIS X 0213 compatibility additions (0xFA30 ~ 0xFA6A)
 *
 *****************************************************************/
WORD tbl_Ideo_Compat [NF_COMPAT][2] = {
{0xF900, 0x8C48}, {0xF901, 0x66F4}, {0xF902, 0x8ECA}, {0xF903, 0x8CC8},
{0xF904, 0x6ED1}, {0xF905, 0x4E32}, {0xF906, 0x53E5}, {0xF907, 0x9F9C},
{0xF908, 0x9F9C}, {0xF909, 0x5951}, {0xF90A, 0x91D1}, {0xF90B, 0x5587},
{0xF90C, 0x5948}, {0xF90D, 0x61F6}, {0xF90E, 0x7669}, {0xF90F, 0x7F85},
{0xF910, 0x863F}, {0xF911, 0x87BA}, {0xF912, 0x88F8}, {0xF913, 0x908F},
{0xF914, 0x6A02}, {0xF915, 0x6D1B}, {0xF916, 0x70D9}, {0xF917, 0x73DE},
{0xF918, 0x843D}, {0xF919, 0x916A}, {0xF91A, 0x99F1}, {0xF91B, 0x4E82},
{0xF91C, 0x5375}, {0xF91D, 0x6B04}, {0xF91E, 0x721B}, {0xF91F, 0x862D},
{0xF920, 0x9E1E}, {0xF921, 0x5D50}, {0xF922, 0x6FEB}, {0xF923, 0x85CD},
{0xF924, 0x8964}, {0xF925, 0x62C9}, {0xF926, 0x81D8}, {0xF927, 0x881F},
{0xF928, 0x5ECA}, {0xF929, 0x6717}, {0xF92A, 0x6D6A}, {0xF92B, 0x72FC},
{0xF92C, 0x90CE}, {0xF92D, 0x4F86}, {0xF92E, 0x51B7}, {0xF92F, 0x52DE},
{0xF930, 0x64C4}, {0xF931, 0x6AD3}, {0xF932, 0x7210}, {0xF933, 0x76E7},
```

```
{0xF934, 0x8001}, {0xF935, 0x8606}, {0xF936, 0x865C}, {0xF937, 0x8DEF},
{0xF938, 0x9732}, {0xF939, 0x9B6F}, {0xF93A, 0x9DFA}, {0xF93B, 0x788C},
{0xF93C, 0x797F}, {0xF93D, 0x7DA0}, {0xF93E, 0x83C9}, {0xF93F, 0x9304},
{0xF940, 0x9E7F}, {0xF941, 0x8AD6}, {0xF942, 0x58DF}, {0xF943, 0x5F04},
{0xF944, 0x7C60}, {0xF945, 0x807E}, {0xF946, 0x7262}, {0xF947, 0x78CA},
{0xF948, 0x8CC2}, {0xF949, 0x96F7}, {0xF94A, 0x58D8}, {0xF94B, 0x5C62},
{0xF94C, 0x6A13}, {0xF94D, 0x6DDA}, {0xF94E, 0x6F0F}, {0xF94F, 0x7D2F},
{0xF950, 0x7E37}, {0xF951, 0x964B}, {0xF952, 0x52D2}, {0xF953, 0x808B},
{0xF954, 0x51DC}, {0xF955, 0x51CC}, {0xF956, 0x7A1C}, {0xF957, 0x7DBE},
{0xF958, 0x83F1}, {0xF959, 0x9675}, {0xF95A, 0x8B80}, {0xF95B, 0x62CF},
{0xF95C, 0x6A02}, {0xF95D, 0x8AFE}, {0xF95E, 0x4E39}, {0xF95F, 0x5BE7},
{0xF960, 0x6012}, {0xF961, 0x7387}, {0xF962, 0x7570}, {0xF963, 0x5317},
{0xF964, 0x78FB}, {0xF965, 0x4FBF}, {0xF966, 0x5FA9}, {0xF967, 0x4E0D},
{0xF968, 0x6CCC}, {0xF969, 0x6578}, {0xF96A, 0x7D22}, {0xF96B, 0x53C3},
{0xF96C, 0x585E}, {0xF96D, 0x7701}, {0xF96E, 0x8449}, {0xF96F, 0x8AAA},
{0xF970, 0x6BBA}, {0xF971, 0x8FB0}, {0xF972, 0x6C88}, {0xF973, 0x62FE},
{0xF974, 0x82E5}, {0xF975, 0x63A0}, {0xF976, 0x7565}, {0xF977, 0x4EAE},
{0xF978, 0x5169}, {0xF979, 0x51C9}, {0xF97A, 0x6881}, {0xF97B, 0x7CE7},
{0xF97C, 0x826F}, {0xF97D, 0x8AD2}, {0xF97E, 0x91CF}, {0xF97F, 0x52F5},
{0xF980, 0x5442}, {0xF981, 0x5973}, {0xF982, 0x5EEC}, {0xF983, 0x65C5},
{0xF984, 0x6FFE}, {0xF985, 0x792A}, {0xF986, 0x95AD}, {0xF987, 0x9A6A},
{0xF988, 0x9E97}, {0xF989, 0x9ECE}, {0xF98A, 0x529B}, {0xF98B, 0x66C6},
{0xF98C, 0x6B77}, {0xF98D, 0x8F62}, {0xF98E, 0x5E74}, {0xF98F, 0x6190},
{0xF990, 0x6200}, {0xF991, 0x649A}, {0xF992, 0x6F23}, {0xF993, 0x7149},
{0xF994, 0x7489}, {0xF995, 0x79CA}, {0xF996, 0x7DF4}, {0xF997, 0x806F},
{0xF998, 0x8F26}, {0xF999, 0x84EE}, {0xF99A, 0x9023}, {0xF99B, 0x934A},
{0xF99C, 0x5217}, {0xF99D, 0x52A3}, {0xF99E, 0x54BD}, {0xF99F, 0x70C8},
{0xF9A0, 0x88C2}, {0xF9A1, 0x8AAA}, {0xF9A2, 0x5EC9}, {0xF9A3, 0x5FF5},
{0xF9A4, 0x637B}, {0xF9A5, 0x6BAE}, {0xF9A6, 0x7C3E}, {0xF9A7, 0x7375},
{0xF9A8, 0x4EE4}, {0xF9A9, 0x56F9}, {0xF9AA, 0x5BE7}, {0xF9AB, 0x5DBA},
{0xF9AC, 0x601C}, {0xF9AD, 0x73B2}, {0xF9AE, 0x7469}, {0xF9AF, 0x7F9A},
{0xF9B0, 0x8046}, {0xF9B1, 0x9234}, {0xF9B2, 0x96F6}, {0xF9B3, 0x9748},
{0xF9B4, 0x9818}, {0xF9B5, 0x4F8B}, {0xF9B6, 0x79AE}, {0xF9B7, 0x91B4},
{0xF9B8, 0x96B8}, {0xF9B9, 0x60E1}, {0xF9BA, 0x4E86}, {0xF9BB, 0x50DA},
{0xF9BC, 0x5BEE}, {0xF9BD, 0x5C3F}, {0xF9BE, 0x6599}, {0xF9BF, 0x6A02},
{0xF9C0, 0x71CE}, {0xF9C1, 0x7642}, {0xF9C2, 0x84FC}, {0xF9C3, 0x907C},
{0xF9C4, 0x9F8D}, {0xF9C5, 0x6688}, {0xF9C6, 0x962E}, {0xF9C7, 0x5289},
{0xF9C8, 0x677B}, {0xF9C9, 0x67F3}, {0xF9CA, 0x6D41}, {0xF9CB, 0x6E9C},
{0xF9CC, 0x7409}, {0xF9CD, 0x7559}, {0xF9CE, 0x786B}, {0xF9CF, 0x7D10},
{0xF9D0, 0x985E}, {0xF9D1, 0x516D}, {0xF9D2, 0x622E}, {0xF9D3, 0x9678},
{0xF9D4, 0x502B}, {0xF9D5, 0x5D19}, {0xF9D6, 0x6DEA}, {0xF9D7, 0x8F2A},
{0xF9D8, 0x5F8B}, {0xF9D9, 0x6144}, {0xF9DA, 0x6817}, {0xF9DB, 0x7387},
{0xF9DC, 0x9686}, {0xF9DD, 0x5229}, {0xF9DE, 0x540F}, {0xF9DF, 0x5C65},
```

```
{0xF9E0, 0x6613}, {0xF9E1, 0x674E}, {0xF9E2, 0x68A8}, {0xF9E3, 0x6CE5},
{0xF9E4, 0x7406}, {0xF9E5, 0x75E2}, {0xF9E6, 0x7F79}, {0xF9E7, 0x88CF},
{0xF9E8, 0x88E1}, {0xF9E9, 0x91CC}, {0xF9EA, 0x96E2}, {0xF9EB, 0x533F},
{0xF9EC, 0x6EBA}, {0xF9ED, 0x541D}, {0xF9EE, 0x71D0}, {0xF9EF, 0x7498},
{0xF9F0, 0x85FA}, {0xF9F1, 0x96A3}, {0xF9F2, 0x9C57}, {0xF9F3, 0x9E9F},
{0xF9F4, 0x6797}, {0xF9F5, 0x6DCB}, {0xF9F6, 0x81E8}, {0xF9F7, 0x7ACB},
{0xF9F8, 0x7B20}, {0xF9F9, 0x7C92}, {0xF9FA, 0x72C0}, {0xF9FB, 0x7099},
{0xF9FC, 0x8B58}, {0xF9FD, 0x4EC0}, {0xF9FE, 0x8336}, {0xF9FF, 0x523A},
{0xFA00, 0x5207}, {0xFA01, 0x5EA6}, {0xFA02, 0x62D3}, {0xFA03, 0x7CD6},
{0xFA04, 0x5B85}, {0xFA05, 0x6D1E}, {0xFA06, 0x66B4}, {0xFA07, 0x8F3B},
{0xFA08, 0x884C}, {0xFA09, 0x964D}, {0xFA0A, 0x898B}, {0xFA0B, 0x5ED3},
{0xFA0C, 0x5140}, {0xFA0D, 0x55C0}, {0xFA0E, 0xFA0E}, {0xFA0F, 0xFA0F},
{0xFA10, 0x585A}, {0xFA11, 0xFA11}, {0xFA12, 0x6674}, {0xFA13, 0xFA13},
{0xFA14, 0xFA14}, {0xFA15, 0x51DE}, {0xFA16, 0x732A}, {0xFA17, 0x76CA},
{0xFA18, 0x793C}, {0xFA19, 0x795E}, {0xFA1A, 0x7965}, {0xFA1B, 0x798F},
{0xFA1C, 0x9756}, {0xFA1D, 0x7CBE}, {0xFA1E, 0x7FBD}, {0xFA1F, 0xFA1F},
{0xFA20, 0x8612}, {0xFA21, 0xFA21}, {0xFA22, 0x8AF8}, {0xFA23, 0xFA23},
{0xFA24, 0xFA24}, {0xFA25, 0x9038}, {0xFA26, 0x90FD}, {0xFA27, 0xFA27},
{0xFA28, 0xFA28}, {0xFA29, 0xFA29}, {0xFA2A, 0x98EF}, {0xFA2B, 0x98FC},
{0xFA2C, 0x9928}, {0xFA2D, 0x9DB4}, {0xFA30, 0x4FAE}, {0xFA31, 0x50E7},
{0xFA32, 0x514D}, {0xFA33, 0x52C9}, {0xFA34, 0x52E4}, {0xFA35, 0x5351},
{0xFA36, 0x559D}, {0xFA37, 0x5606}, {0xFA38, 0x5668}, {0xFA39, 0x5840},
{0xFA3A, 0x58A8}, {0xFA3B, 0x5C64}, {0xFA3C, 0x5C6E}, {0xFA3D, 0x6094},
{0xFA3E, 0x6168}, {0xFA3F, 0x618E}, {0xFA40, 0x61F2}, {0xFA41, 0x654F},
{0xFA42, 0x65E2}, {0xFA43, 0x6691}, {0xFA44, 0x6885}, {0xFA45, 0x6D77},
{0xFA46, 0x6E1A}, {0xFA47, 0x6F22}, {0xFA48, 0x716E}, {0xFA49, 0x722B},
{0xFA4A, 0x7422}, {0xFA4B, 0x7891}, {0xFA4C, 0x793E}, {0xFA4D, 0x7949},
{0xFA4E, 0x7948}, {0xFA4F, 0x7950}, {0xFA50, 0x7956}, {0xFA51, 0x795D},
{0xFA52, 0x798D}, {0xFA53, 0x798E}, {0xFA54, 0x7A40}, {0xFA55, 0x7A81},
{0xFA56, 0x7BC0}, {0xFA57, 0x7DF4}, {0xFA58, 0x7E09}, {0xFA59, 0x7E41},
{0xFA5A, 0x7F72}, {0xFA5B, 0x8005}, {0xFA5C, 0x81ED}, {0xFA5D, 0x8279},
{0xFA5E, 0x8279}, {0xFA5F, 0x8457}, {0xFA60, 0x8910}, {0xFA61, 0x8996},
{0xFA62, 0x8B01}, {0xFA63, 0x8B39}, {0xFA64, 0x8CD3}, {0xFA65, 0x8D08},
{0xFA66, 0x8FB6}, {0xFA67, 0x9038}, {0xFA68, 0x96E3}, {0xFA69, 0x97FF},
{0xFA6A, 0x983B}
};

int main (int argc, char *argv[])
{
        int i, n_cnt = 0;
        char str[128]= "";

        // Begin Of Mark
```

```
        WORD BOM = 0xFEFF;
        FILE *fp1, *fp2;

        // Open two files
        fp1 = fopen ("CJK_Compat_Ideo.txt", "w+b");
        fp2 = fopen ("CJK_Ideo.txt", "w+b");

        // If it cannot open files, fails
        if (fp1 == NULL || fp2 == NULL ) return 1;

        fwrite ( (WORD *) &BOM, (size_t) sizeof(WORD), (size_t) 1, fp1);
        fwrite ( (WORD *) &BOM, (size_t) sizeof(WORD), (size_t) 1, fp2);

        while ( n_cnt < 361 ) {
        // CJK Compatibility Ideographs Area 출력
        fwrite ((WORD *) &tbl_Ideo_Compat[n_cnt][0],
                (size_t) sizeof(WORD), (size_t) 1, fp1);
        // CJK Compatibility Ideographs Area에 해당하는 CJK Ideographs Area 출력
        fwrite ((WORD *) &tbl_Ideo_Compat[n_cnt][1],
                (size_t) sizeof(WORD), (size_t) 1, fp2);
        n_cnt++;
        }
        // close two files
        fclose(fp1);
        fclose(fp2);
        return 0;
}
```